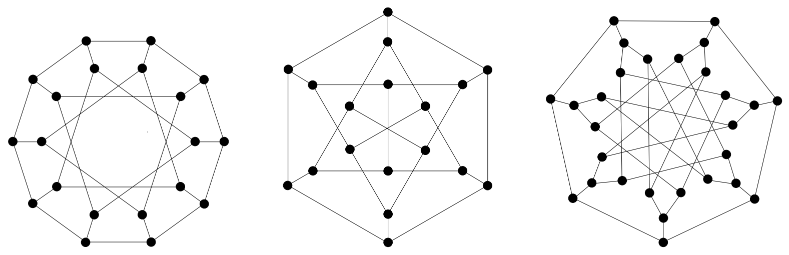
Isomorph-free exhaustive generation algorithms for association schemes

Isomorfvrije exhaustieve generatiealgoritmen voor associatieschema's

Jan Degraer



Promotor: prof. dr. dr. K. Coolsaet

Proefschrift ingediend tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica

Vakgroep Toegepaste Wiskunde en Informatica
Voorzitter: prof. dr. G. Vanden Berghe
Faculteit Wetenschappen
Academiejaar 2006-2007

UNIVERSITEIT
GENT

To Lien with all my love.

Acknowledgements

I may not have gone where I intended to go, but I think I have ended up where I needed to be. [D. ADAMS]

First and foremost, my thanks go to my promotor Prof. Kris Coolsaet for following my research with interest and for giving me support and guidance countless times. During the last six years I have learnt a lot from your profound knowledge in computer science and mathematics and your talent to simplify anything that looks difficult. Your confidence in my abilities has been a stimulating challenge for me. Thanks Kris !

Further I would like to thank all my colleagues and ex-colleagues in the department for a friendly and sociable atmosphere. It is nearly impossible to list all these people here. However, the following people deserve a special mention. Thank you Veerle and Gunnar for being there for me over the past years. Thank you Adriaan, Guy and Reza – my office mates – for your friendship.

I am grateful to my parents for their trust and love. And last but certainly not least, I would like to thank Lien. However, I just do not know how to thank you for all your support, patience and love, especially during the final months of writing this dissertation.

Jan Degraer
march 2007

Inhoudsopgave

1 Introduction

”Not everything that counts can be counted, and not everything that can be counted counts.”[A. EINSTEIN]

Combinatorics is a branch of discrete mathematics dealing with finite numbers of objects satisfying a certain set of constraints. A wide spectrum of combinatorial objects are of interest to combinatorists, ranging from rather basic objects such as all the subsets of a finite set, permutations, partitions, trees, . . . to more complex combinatorial objects such as graphs [9, 12], designs [26, 65], geometries [60], posets [11] . . . whether satisfying various structural properties or not. In the study of such combinatorial objects the following principal problems arise frequently [65]:

existence problem Does a combinatorial object that satisfies a given set of constraints exist?

counting problem Given a set of constraints, count, up to some notion of *isomorphism*, the number of combinatorial objects meeting these constraints.

classification problem Given a set of constraints, describe, up to some notion of *isomorphism*, all combinatorial objects meeting these constraints or establish non-existence.

Probably the most fundamental problem is the problem of existence. In general, it is straightforward to verify that a combinatorial object meets the required constraints, however, an explicit construction of such an object is often difficult to realize. In the case of counting and classification problems it is customary to suppose that the combinatorial objects under consideration are subject to some relation "is isomorphic to", which partitions these combinatorial objects into equivalence classes. In this way, exactly one representative from each equivalence class, or isomorphism class, is counted or described and combinatorial objects within a given equivalence class are considered mathematically identical, or so-called isomorphic. In general, two combinatorial objects can be regarded as isomorphic, if the one can be acquired from the other, and vice versa, by renaming or reordering (or both) the elementary objects – such as elements or subsets of a finite set – of which these combinatorial objects are composed.

The following well-known problem posed by the Reverend Thomas Penyngton Kirkman in 1850 in a popular magazine called *The Lady's and Gentleman's Diary* might serve to clarify the terms combinatorial object, combinatorial classification and isomorphism.

Kirkman's schoolgirl problem: Fifteen young ladies in a school walk out three abreast for seven day in succession: it is required to arrange them daily, so that no two walk twice abreast.

In order to classify all such week schedules, any two schedules are considered identical (isomorphic) when one such schedule can be obtained from the other by renaming the schoolgirls and reordering the days. Taking into account this particular definition of isomorphism, the valid week schedules turn out to be partitioned into exactly seven isomorphism classes. Representatives of each such class are shown in Table 1.1, where letters represent schoolgirls, groups of three schoolgirls are listed vertically.

When we look at the 7 distinct week schedules of Kirkman's schoolgirl problem, it is rather easy to verify that each week schedule is correct. However, there are 5 040 ways to reorder the days and 1 307 674 368 000 ways to rename the schoolgirls. Taking into account this vast amount of possibilities, verifying whether or not these 7 solutions represent exactly one solution from every isomorphism

Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
adefg	abcdf	abcdg	abcfg	abcde	abcef	abceg
bjhik	hemkj	jmehi	dlikh	fhlik	lidjh	ndhij
cnmol	ignol	kofln	enjmo	gjomn	mkgon	ofklm
adefg	abcdf	abcdg	abcfg	abcde	abceg	abcef
bjhik	hemkj	jmehi	dilhj	flhij	ldikh	nhdik
cnmol	ignol	kofln	ekonm	gnkmo	mfjno	ojglm
adefg	abceg	abcdf	abcfg	abcde	abcef	abcdg
bjikh	hdlkj	jemhi	dlihk	fmhij	lidhj	nheki
comln	ifonm	kgnlo	enjmo	goknl	mkgon	ojfml
adefg	abceg	abcdf	abcfg	abcde	abcdg	abcef
bjikh	hdlkj	jemhi	dmhji	flikh	lheik	nidjh
comln	ifonm	kgnlo	eoknl	gnmjo	mjfno	okglm
abcfg	abcde	abceg	abcdf	adefg	abcdg	abcef
dihmj	fhikl	ldkih	nejih	bhkji	hmejk	jldhi
eklno	gjnmo	mfojn	ogmlk	conlm	iofnl	kngmo
abcfg	abcde	abceg	abcdf	adefg	abcef	abcdg
dihmj	fhikl	ldkih	nejih	bjhik	hmdkj	jlehi
eklno	gjnmo	mfojn	ogmlk	cnmol	iognl	knfom
adefg	abcfg	abcde	abcdf	abcdg	abcef	abceg
bijlh	dijhk	fhimk	heljk	jmehi	lkdhi	ndhij
cknom	elmno	gjonl	ignom	kofln	mngoj	ofkml

Tabel 1.1: The 7 distinct solutions to Kirkman's schoolgirl problem [27].

class involved, is certainly far less straightforward. In these cases, where classification results can not be established by means of combinatorial arguments, combinatorial classification by means of manual calculations turns out to be a onerous, tedious and error-prone task.

Nowadays combinatorial classification is inextricably bound up with computers and classification algorithms. Algorithms have become essential tools, in successfully solving both existence problems [38, 45, 81, 84, 90] as well as classification problems [37, 66, 79, 82, 95] for a wide variety of combinatorial objects. Combinatorial objects are both finite and discrete structures which makes corresponding existence and classification problems suitable to be tackled with local and exhaustive search algorithms. Local search algorithms, such as hill climbing, simulated annealing, ... are commonly used to solve existence problems. Such methods do not guarantee to find a solution even if one or more solutions exist, however, in practice these methods turn out to be efficient.

Exhaustive search algorithms on the other hand are often used to solve classification problems or even to establish non-existence – probably one of the most well-known classification results is the nonexistence of the projective plane of order 10 [68] (see also [65, chapter 12]). Such exhaustive methods examine all candidate solutions within a given search space, that is why they guarantee to find all solutions if such solutions exist – under the assumption that one is willing to wait long enough. Probably the most well-known exhaustive search method is backtracking [49], in which partial solutions are recursively extended and collapsed one step at a time until a solution is encountered. Backtracking can be seen as a depth first traversal of a search tree, in which nodes correspond to partial solutions and branches correspond to the systematic extension of a partial solution, or in the opposite direction, the systematic collapse of a partial solution.

In this text we concentrate on algorithms for classifying combinatorial objects up to isomorphism, or so-called isomorph-free exhaustive generation algorithms. The combinatorial objects under consideration are association schemes [1, 47] and in particular strongly regular [13, 14] and distance regular graphs [12]. Our ambition is dual. The emphasis is primarily from a *computer science perspective*, but also the mathematical results we have obtained are of independent interest.

We try to design, improve and study such classification algorithms for hard combinatorial problems. These algorithms essentially require a recursive traversal of a tree-like search space which exhibits an exponential behaviour. Typically, various intelligent pruning methods are used to limit the search space, and thus at the same time, to control the execution time of the classification algorithm.

Basically, there are two distinct types of pruning methods. On the one hand, we may prune branches of the recursion tree which correspond to partial solutions for which it can be established that they cannot possibly be extended to a full solution satisfying the required constraints. In practice, establishing the non-extensibility of a partial solution is based on the mathematical properties that are specific to the combinatorial objects at hand. On the other hand, we need methods for determining isomorphic (partial) solutions so as to detect symmetry in, and eliminate symmetry from the tree-like search space. More precisely, we can prune branches of the recursion tree which correspond to partial solutions for which it can be proved that they are isomorphic to partial solutions we have already considered earlier during search. Note that these methods are also used to guarantee that exactly one representative of each isomorphism class is listed as result of the classification algorithm.

Despite the usage of clever, well-designed pruning methods, the search space associated with the classification of certain combinatorial objects can easily exceed the capacity of the finite computing resources available. Two fundamentally distinct problems – or even the combination of both – may underlie this phenomenon. A first type of problem which may occur is that there are far more non-isomorphic combinatorial objects than we can process with the computer resources at our disposal. For example, the classification of the Steiner triple systems[1] of order 15 (abbreviated by STS(v) where v denotes the order) – one of the most renowned paper-and-pencil calculations – resulted in the discovery of 80 pairwise non-isomorphic designs [25]. For STS(19), the usage of approximately two years of CPU time – on a 500-MHz computer – was needed in order to obtain 11 084 874 829 pairwise non-isomorphic designs [63]. For STS(21), a complete classification is currently out of reach [64]. Using the same classification algorithm as for STS(19), an estimation of hundreds of thousands CPU-years is given in [63]. Because the aforementioned numbers of pairwise non-isomorphic STS(15), STS(19) and STS(21) are lower bounds on the number of pairwise non-isomorphic strongly regular graphs with parameters

[1]For lower and upperbounds on the number of non-isomorphic Steiner triple systems we refer to [107].

$(35, 16, 6, 8)$, $(57, 24, 11, 9)$ and $(70, 27, 12, 9)$ respectively, the same problem is applicable to the classification of at least some of the combinatorial objects under consideration in this text.

When designing a classification algorithm, we face the challenge to discover and prove mathematical properties of the combinatorial objects at hand which might contribute to a great extent in reducing the corresponding search space. A second type of problem which may occur is that despite a clever usage of these mathematical properties, the classification algorithm may still end up in traversing large parts of the search space which do not contain a single solution. It was long an open problem whether a $2-(22, 8, 4)$ block design exists [3, 57, 73, 85]. Even with a highly sophisticated generation algorithm, recently still more than 263 CPU years – on a 2-GHz computer – were used to establish that no $2-(22, 8, 4)$ block designs exist [4].

From a more *mathematical perspective*, results obtained using classification algorithms are an objective by itself. Classification results are of interest to mathematicians since a complete catalogue of such objects may provide a better comprehension of their structure or, even more, of a larger family of combinatorial objects to which they belong. Furthermore, catalogues of combinatorial objects can be used for testing existing conjectures, formulating new conjectures or to find objects within this catalogue which satisfy certain mathematical properties of interest. For example, the classification of the strongly regular $(45, 12, 3, 3)$ graphs [22] – one of the results of this thesis – was used in [42] to determine all primitive strongly regular graphs with chromatic number equal to 5 and in [39] to conjecture that the isomorphism classes of strongly regular graphs are characterized by the spectrum of certain matrices associated with each graph. As a second example, the classification of the Perkel graph – another result of this thesis – was used in [102, 103] to determine which distance-regular graphs are characterized by their graph spectrum. A catalogue of combinatorial objects can also serve as a starting point in the classification of related combinatorial structures. For example, the strongly regular $(45, 12, 3, 3)$ graphs were used in [61] to classify all primitive non-symmetric 3-class association schemes on at most 100 vertices and applied in [31] to construct new symmetric 3-class association schemes.

The organization of the remainder of this text is as follows. Chapter 2 outlines the theoretical background on association schemes, strongly regular graphs and distance regular graphs. In Chapter 3 we introduce, in a group-theoretic framework, several generic algorithms used for exhaustively generating a collection of combinatorial objects up to isomorphism. Specific algorithms for association schemes are discussed in Chapters 4 and 5. More precisely, Chapter 4 focuses on specific methods to generate at least one representative from every isomorphism class, while Chapter 5 emphasizes specific methods to remove isomorphic association schemes from consideration so as to obtain isomorph-free generation. Finally we present in Chapter 6 several case studies together with new complete classifications results on association schemes, strongly regular and distance regular graphs obtained using (variants of) the generation algorithms described in Chapters 4 and 5. Several of these classification results appear in [21, 22, 30, 31]. Attacking computational hard classification problems sometimes requires to tackle such problems as individual instances, because no algorithm that applies to a general class, will be feasible for some particular instances. If so, we outline for each such instance the principal modifications to the general generation algorithm.

2 Association schemes

”Someone told me that each equation I included in the book would halve the sales.”[S. HAWKING]

In this chapter we give an introduction to the theory of association schemes, placing emphasis on the concepts that will be necessary for the remainder of this text. At the same time, the concepts of distance regular and strongly regular graphs and their connection with association schemes are introduced.

Association schemes play an important role in the field of combinatorics. They have their roots in the statistical design of experiments [7] and in the study of groups acting on finite sets [59]. Besides, associations schemes are used in coding theory [32], design theory and graph theory. In terms of graphs, association schemes can be regarded as colourings of the edges of the complete graph satisfying nice regularity conditions.

Many books are devoted to the theory of association schemes and serve as basis for parts of this introduction. For a more elaborate introduction to the theory of association schemes, the reader might wish to consult [1, 2, 12, 47].

2.1 Graphs

First we recall some general concepts from graph theory. For an extensive overview of graph theory we refer to [28, 36, 105].

Definition 2.1.1. A *graph* G is an ordered pair (V, E), where V is a finite set of *vertices* and E is a finite set of two-element subsets of vertices, called *edges*. $\diamond$

The vertex set of a graph G is denoted by $V(G)$, its edge set by $E(G)$. We shall write V instead of $V(G)$ and E instead of $E(G)$ whenever the graph G is clear from context. The cardinality of $V(G)$ is called the *order* of G while the cardinality of $E(G)$ is called the *size* of G. An edge $\{x, y\}$, commonly written as xy or yx, *joins* the vertices x and y. Two vertices are *adjacent* if they are joined by an edge. We shall write $x \sim y$ to indicate that two vertices are adjacent (and distinct) and $x \not\sim y$ to indicate that they are not adjacent (and distinct). A vertex x is *incident* with an edge e if $x \in e$. The *neighbourhood* $N_G(x) = N(x)$ of a vertex x is the set of all vertices adjacent to x. The *degree* or *valency* $d(x)$ of a vertex x is the number of vertices adjacent to x (hence $d(x) = |N(x)|$).

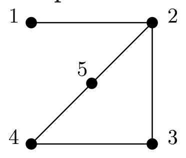
The *complement* $\overline{G}$ of G is the graph with the same vertex set V such that two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. If all vertices of G are pairwise adjacent, then G is *complete*. A complete graph on n vertices is denoted by K_n. The complement of a complete graph is an *empty* graph. A *path* is a non-empty graph G with $V = \{x_1, x_2, \ldots, x_k\}$ and $E = \{\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{k-1}, x_k\}\}$. The number of edges of a path is its *length*. A non-empty graph G is *connected* if any two vertices are linked by a path. A connected graph G is a *tree* if the path linking any two vertices is unique.

The *distance* $d(x, y)$ in a graph G of two of its vertices is the length of the shortest path linking x and y. The greatest distance between any two vertices of a graph G is the *diameter* of G. The *i-th neighbourhood* $N_i(x)$ of a vertex x is the set of all vertices that lie at distance i from x (hence $N(x) = \mathbb{N}_1(x)$). The *distance-k* graph G_k, $0 \le k \le d$, of a connected graph G of diameter d, is the graph with the same vertex set V such that any two vertices are adjacent whenever they are at distance k in G. Hence $N_i(x) = N_{G_i}(x)$. A graph G of diameter d is an *antipodal* graph if, for any vertex $x \in V$, the set $\{x\} \cup N_d(x)$ consists of vertices which are mutually at distance d. Hence, there exists a partition of the vertex set into classes such that any two vertices are in the same class if and only if they are at distance d.

If all the vertices of G have the same degree k, then G is called *regular* of degree k or *k-regular*. A 3-regular graph is also called a *cubic* graph. A $\mathrm{reg}(k, n)$ is a regular graph of degree k on n vertices. A graph G is *bipartite* if its vertex set V can be partitioned into two partite sets V_1 and V_2 such that every edge of E joins a vertex of V_1 to a vertex of V_2. A *complete bipartite* graph G is a bipartite graph with partite sets V_1 and V_2 such that all vertex pairs with one vertex in V_1 and the other in V_2 are joined by an edge.

Graphs can be represented in several ways. Most commonly a graph is depicted by drawing a point for each vertex, where two points are connected with a line if the corresponding vertices form an edge.

Example 2.1. The graph G with $E(G) = \{\{1,2\},\{2,3\},\{2,5\},\{3,4\},\{4,5\}\}$ and $V(G) = \{1,2,3,4,5\}$ may be represented by



•

A graph can also be described by means of its *adjacency* matrix.

Definition 2.1.2. Let $G = (V, E)$ with $V = \{x_1, \dots, x_n$, then the corresponding *adjacency matrix* of G is the $n \times n$ matrix $A = A(G)$ where for all $i,\ j \in \{1, \dots, n\}$ its matrix entry A_{ij} at row i and column j is defined as follows:

$$A_{ij} = \begin{cases} 1 & \text{if } x_i \sim x_j \\ 0 & \text{otherwise.} \end{cases}$$

◇

Hence the adjacency matrix of a graph is a symmetric $(0, 1)$ matrix having entries equal to zero along the main diagonal. Note that the adjacency matrix of G depends on the chosen ordering of its vertices. (When $V = \{1, \dots, n\}$, we usually choose $x_i = i$.)

Example 2.2. The adjacency matrix A of the graph in Example 2.1 is given by

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

•

The *spectrum* of a graph G is the multiset

$$\operatorname{spec}(G) = \{\lambda_0, \ldots, \lambda_0, \lambda_1, \ldots, \lambda_1, \ldots\}$$

of all eigenvalues of $A(G)$. (Note that these eigenvalues are independent of the choice of the ordering of the vertices of G.) We usually write

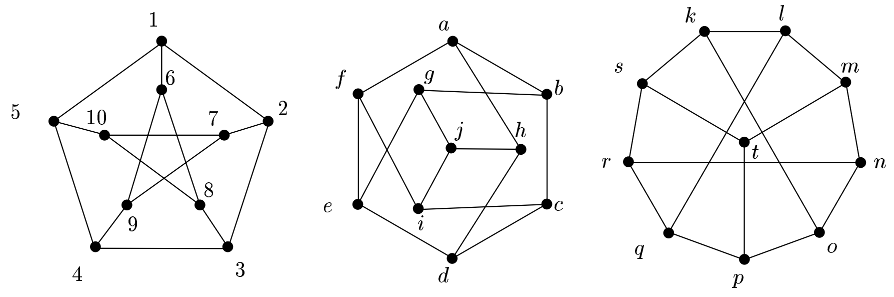
$$\operatorname{spec}(G) = \{(\lambda_0)^{m_0}, (\lambda_1)^{m_1}, \ldots\}$$

where the superscripts denote the multiplicities of the corresponding eigenvalues.

Some graphs have the same structure, differing only in the way their vertices and edges are labeled. To formalize the meaning of "the same structure", we introduce the concept of *isomorphism*.

Definition 2.1.3. A graph G and H are *isomorphic*, written $G \cong H$, if there exists a bijection $\varphi : V(G) \to V(H)$ such that, for all $x,\ y \in V(G)$, we have $xy \in E(G)$ if and only if $\varphi(x)\varphi(y) \in E(H)$. Such a map φ is called an *isomorphism*; if $G = H$ then it is called an *automorphism*. ◇

Example 2.3. Three isomorphic graphs are shown below.



The map φ such that $\varphi(1) = a$, $\varphi(2) = b$, $\varphi(3) = g$, $\varphi(4) = e$, $\varphi(5) = f$, $\varphi(6) = h$, $\varphi(7) = c$, $\varphi(8) = j$, $\varphi(9) = d$, $\varphi(10) = i$, is an isomorphism of the first graph onto the second graph.

The map ϕ such that $\phi(1) = k$, $\phi(2) = l$, $\phi(3) = m$, $\phi(4) = n$, $\phi(5) = o$, $\phi(6) = s$, $\phi(7) = q$, $\phi(8) = t$, $\phi(9) = r$, $\phi(10) = p$, is an isomorphism of the first graph onto the third graph.

Finally, the map σ such that $\sigma(a) = k$, $\sigma(b) = l$, $\sigma(c) = q$, $\sigma(d) = r$, $\sigma(e) = n$, $\sigma(f) = o$, $\sigma(g) = m$, $\sigma(h) = s$, $\sigma(i) = p$, $\sigma(j) = t$, is an isomorphism of the second graph onto the third graph. ●

Definition 2.1.4. A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. ◇

A subgraph H of G is *spanning* if $V(H) = V(G)$. An *induced* subgraph H of G is *induced* by $V' \subseteq V(G)$ if it has V' as its vertex set and if it contains all edges xy of $E(G)$ such that $x, y \in V'$. A *clique* in a graph G is a set of vertices whose induced subgraph is a complete graph. The *clique number* of a graph G is the number of vertices in the largest clique of G.

2.2 Association schemes

In this section we successively describe association schemes in terms of relations, graphs and matrices. We also provide some well-known examples.

Definition 2.2.1. A *d-class association scheme* Ω on a finite set V is an ordered set $\{R_0, R_1, \ldots, R_d\}$ of relations on the set V which satisfies the following axioms:

1. $\{R_0, R_1, \ldots, R_d\}$ is a *partition* of $V \times V$.
2. R_0 is the *identity relation*, i.e., $(x, y) \in R_0$ if and only if $x = y$, whenever $x, y \in V$.
3. Every relation R_i is *symmetric*, i.e., if $(x, y) \in R_i$ then also $(y, x) \in R_i$, for every $x, y \in V$.

4. Let $0 \leq i, j, l \leq d$. Let $x, y \in V$ such that $(x, y) \in \mathrm{R}_l$, then the number
$$p_{ij}^{l} = |\{z \in V : (x, z) \in \mathrm{R}_i \text{ and } (z, y) \in \mathrm{R}_j\}|$$
only depends on i, j and l.

$\diamond$

The relations $\mathrm{R}_0, \mathrm{R}_1, \dots, \mathrm{R}_d$ are called the *associate classes* of the scheme; two elements $x, y \in V$ are i-th associates if $(x, y) \in R_i$. We use the notation $x\mathrm{R}_i y$ to indicate that $(x, y) \in \mathrm{R}_i$. The numbers p_{ij}^{l} are called the *intersection numbers* of Ω. It is common practice to write the intersection numbers p_{ij}^{l} as entries of the so-called *intersection matrices* $L_0, L_1, \dots, L_d$ where

$$(L_i)_{lj} = p_{ij}^{l} \tag{2.1}$$

with $0 \leq i,\ j,\ l \leq d$. Note that $L_0 = 1$, the identity matrix. Define $v = |V|$, and $k_i = p_{ii}^{0}$. The *valency* or *degree* k_i denotes the number of elements $y \in V$ in relation R_i to a fixed element $x \in V$. This number also does not depend on the choice of x. The set of all d-class association schemes on $V = \{1, \dots, n\}$ with intersection matrices $L_0, L_1, \dots, L_d$ shall be denoted by $\mathcal{X}_{L_0 \dots L_d}$. (We assume that $n = \sum_{k=0}^{d} p_{kk}^{0}$.)

A d-class association scheme Ω on a finite set V is called *imprimitive* if some union of relations is an equivalence relation distinct from R_0 and $V \times V$.

Example 2.4. Define the n-class *Hamming scheme* $H(n, q)$ as follows: the elements are the q^n q-tuples over an alphabet of cardinality q, two elements are i-th associates if they differ in i positions. Hence for $q = 2$, this is the n-dimensional hypercube, where two elements x, y are i-th associates if they are at distance $i = d(x, y)$. $\bullet$

Example 2.5. Define the q-class *Johnson scheme* $J(n, q)$ as follows: the elements are the $\binom{n}{q}$ q-subsets of a fixed n-set, two elements are i-th associates if they intersect in exactly $q - i$ points. $\bullet$

Example 2.6. Define the 3-class *Rectangular scheme* $R(n, m)$ with $m, n \geq 2$ as follows: the elements are the $n\,m$ entries of an $n \times m$ array, two elements are 1-st associates if they are in the same row, 2-th associates if they are in the same column and 3-th associates if they are in different rows and columns. $\bullet$

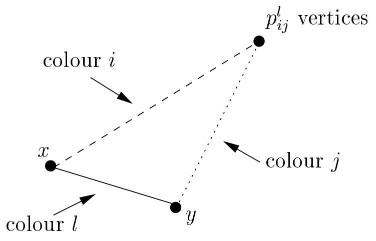
We can also consider d-class association schemes in terms of *graphs*. Let $i \in \{1, \dots, d\}$. With every relation R_i we may associate a simple graph $G_i = (V, R_i)$ of order v. Two vertices $x, y \in V$ are adjacent in G_i if and only if xR_iy. Clearly G_i must be a regular graph of degree k_i. Since the ordered set of relations $\{\mathrm{R}_0, \mathrm{R}_1, \dots, \mathrm{R}_d\}$ is a *partition* of $V \times V$, we may consider association schemes as partitions of the complete graph K_v into d regular subgraphs, which by the fourth defining axiom are interrelated in a specific way.

If for every graph G_i with $i \in \{1, \dots, d\}$, we colour all of its edges with colour i, then we may also consider a d-class association scheme as a colouring of the complete graph K_v with d colours. In particular the fourth defining axiom induces an additional condition about the number of triangles of various types through an edge of a specified colour. Consider two distinct vertices $x,\ y \in V$ with edge xy of colour l, then the number

$$p^l_{ij} = |\{z \in V : \text{edge } xz \text{ is of colour } i \text{ and edge } zy \text{ is of colour } j\}|$$

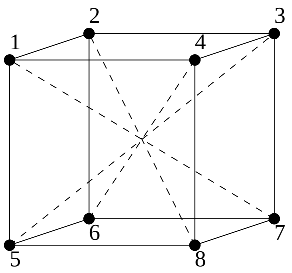
only depends on i, j and l with $0 < i,\ j,\ l \le d$.

This condition states that if we fix two distinct vertices $x, y \in V$, and colours i, j and l, then the number of *triangles* with as edges the l-coloured edge xy, an i-coloured edge through x and a j-coloured edge through y is exactly p^l_{ij}, irrespective of the choice of the vertex x and the vertex y.



Moreover, if we attach a loop to all vertices of V and colour each loop with the colour 0, then we find that $k_0 = 1$, $k_i = p^0_{ii}$ and $p^i_{0j} = p^i_{j0} = \delta_{ij}$. This means that every vertex $x \in V$ is contained in exactly k_i edges of colour i.

Example 2.7. Let $V = \{1, \dots, 8\}$ be the set of eight vertices of the cube shown below.



Colour the edges of the cube with colour 1 (solid lines), the main diagonals with colour 2 (dashed lines) and the face diagonals with colour 3 (not shown). Then we find a 3-class association scheme with valencies $k_1 = 3$, $k_2 = 1$ and $k_3 = 3$, and intersection matrices

$$L_1 = \begin{pmatrix} 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 3 \\ 0 & 2 & 1 & 0 \end{pmatrix} \quad L_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad L_3 = \begin{pmatrix} 0 & 0 & 0 & 3 \\ 0 & 2 & 1 & 0 \\ 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix}$$

•

Finally we can consider association schemes in terms of *matrices*. With every relation R_i, $i \in \{0, \ldots, d\}$, we may associate a $(0,1)$-matrix A_i of dimension $v \times v$ as follows: rows and columns of A_i are indexed by the elements of V and for every $x, y \in V$ the matrix entry $(A_i)_{xy}$ is defined by

$$(A_i)_{xy} = \begin{cases} 1 & \text{when } x\mathrm{R}_i y, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

Clearly, the $(0,1)$-matrix A_i is the *adjacency matrix* of the k_i-regular graph $G_i = (V, \mathrm{R}_i)$. In terms of the $(0,1)$-matrices $A_0, A_1, \ldots, A_d$, an association scheme Ω can be represented by its square symmetric *relation matrix* M_Ω of dimension n such that

$$M_\Omega = \sum_{k=0}^{d} k\, A_k \tag{2.3}$$

In other words, the matrix entry $(M_\Omega)_{xy} = i$ if and only if $(x, y) \in R_i$.

Example 2.8. Consider the 3-class association scheme Ω on $V = \{1, \ldots, 8\}$ with intersection matrices L_1, L_2, L_3 are defined as in Example 2.7. Then the corresponding relation matrix M_Ω is shown below.

$$M_\Omega = \begin{pmatrix} 0 & 1 & 3 & 1 & 1 & 3 & 2 & 3 \\ 1 & 0 & 1 & 3 & 3 & 1 & 3 & 2 \\ 3 & 1 & 0 & 1 & 2 & 3 & 1 & 3 \\ 1 & 3 & 1 & 0 & 3 & 2 & 3 & 1 \\ 1 & 3 & 2 & 3 & 0 & 1 & 3 & 1 \\ 3 & 1 & 3 & 2 & 1 & 0 & 1 & 3 \\ 2 & 3 & 1 & 3 & 3 & 1 & 0 & 1 \\ 3 & 2 & 3 & 1 & 1 & 3 & 1 & 0 \end{pmatrix}$$

●

In terms of the $(0,1)$-matrices $A_0, A_1, \ldots, A_d$ the four defining axioms of a d-class association scheme translate to the following four statements:

$$\sum_{l=0}^{d} A_l = J, \quad A_0 = 1, \quad A_i = A_i^T \quad \text{and} \quad A_iA_j = \sum_{l=0}^{d} p_{ij}^l A_l \tag{2.4}$$

with $0 \leq i, j \leq d$ and where 1 denotes the $v \times v$ identity matrix, J the $v \times v$ all-one matrix and A^T is the transpose of A. The first three statements are straightforward. The fourth statement is valid as for every $x, y \in V$, the matrix entry $(A_iA_j)_{xy}$ is equal to p_{ij}^l whenever xR_ly with $0 \leq i, j \leq d$.

Lemma 2.2.2. *The matrices $A_0, A_1, \ldots, A_d$ of a d-class association scheme satisfy $A_iA_j = A_jA_i$ with $0 \leq i, j \leq d$.*

Proof : By the third and fourth statement of (2.4) we have

$$A_iA_j = A_i^TA_j^T = (A_jA_i)^T = \sum_{l=0}^{d} p_{ji}^l A_l^T = \sum_{l=0}^{d} p_{ji}^l A_l = A_jA_i.$$

■

Lemma 2.2.3. *Let $0 \leq i, j, l \leq d$. Then the intersection numbers p_{ij}^l and valencies k_i of a d-class association scheme satisfy the following statements:*

1. $p_{0j}^l = \delta_{jl}$, $p_{ij}^0 = \delta_{ij}k_j$, $p_{ij}^l = p_{ji}^l$,
2. $\sum_{j=0}^{d} p_{ij}^l = k_i$, $\sum_{i=0}^{d} k_i = v$,

3. $p^l_{ij}k_l = p^j_{il}k_j$,

4. $\sum_{l=0}^d p^l_{ij}p^m_{ln} = \sum_{l=0}^d p^m_{il}p^l_{jn}$.

Proof : Statements 1–3 are straightforward. The fourth statement is valid as the expressions at both sides count quadruples (u, v, x, y) with $u\mathrm{R}_i v$, $v\mathrm{R}_j x$, $x\mathrm{R}_n y$, for a fixed pair (u, y) with $u\mathrm{R}_m y$. ■

2.3 Distance regular and strongly regular graphs

In this section we introduce the concept of distance regular and strongly regular graphs and their connection with association schemes. For an extensive overview of the theory of distance regular graphs we refer to [12].

Definition 2.3.1. A connected graph $G = (V, E)$ with diameter d is called *distance regular* if it is regular of valency k, and there are positive integers b_i and c_i with $0 \leq i \leq d$ such that for any two vertices $x, y \in V$ at distance $i = d(x, y)$, there are precisely c_i neighbours of y in $N_{i-1}(x)$ and b_i neighbours of y in $N_{i+1}(x)$. ◇

The sequence

$$\{b_0, b_1, \ldots, b_{d-1}; c_1, c_2, \ldots, c_d\} \tag{2.5}$$

is called the *intersection array* of G. We also write

$$a_i = k - b_i - c_i, \tag{2.6}$$

for the number of neighbours of y in $N_i(x)$, The numbers c_i, b_i and a_i are called the *intersection numbers* of the distance regular graph G. Note that $b_0 = k$, $b_d = c_0 = 0$ and $c_1 = 1$.

By counting all edges yz with $y, z \in V$, $d(x, y) = i$ and $d(x, z) = i + 1$ we see that $N_i(x)$ contains k_i vertices with $0 \leq i < d$, where

$$k_0 = 1, \quad k_1 = k, \quad k_{i+1} = k_i b_i / c_{i+1}. \tag{2.7}$$

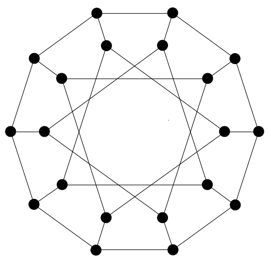
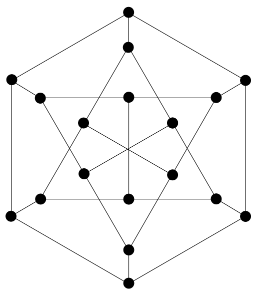
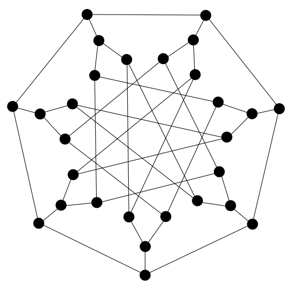
We have $|V| = 1 + k_1 + \ldots + k_d$.

A distance regular graph G of diameter d is called *imprimitive* when for some i, $0 \leq i \leq d$ the distance graph G_i is disconnected. In [100], Smith proved that an imprimitive distance regular graph of valency $k > 2$ is either *bipartite* (with distance graph G_2 disconnected) or *antipodal* (graphs such that their distance graph G_d is an equivalence relation) or both.

Example 2.9. Simple examples of distance regular graphs are the *polygons*. A polygon has intersection array $\{2, 1, \ldots, 1; 1, \ldots, 1, c_d\}$, where $c_d = 2$ for the $2d$-gon and $c_d = 1$ for the $(2d+1)$-gon. •

Example 2.10. A Platonic graph corresponds to the skeleton of a *Platonic solid*. The five Platonic graphs, that is, the tetrahedron, octahedron, cube, icosahedron and dodecahedron are distance regular graphs with intersection arrays given by $\{3; 1\}$, $\{4, 1; 1, 4\}$, $\{3, 2, 1; 1, 2, 3\}$, $\{5, 2, 1; 1, 2, 5\}$ and $\{3, 2, 1, 1, 1; 1, 1, 1, 2, 3\}$, respectively. •

Example 2.11. The Desargues graph, the Pappus graph and the Coxeter graph are shown below.

All three graphs are cubic distance regular graphs with intersection arrays given by $\{3, 2, 2, 1, 1; 1, 1, 2, 2, 3\}$, $\{3, 2, 2, 1; 1, 1, 2, 3\}$ and $\{3, 2, 2, 1; 1, 1, 1, 2\}$, respectively. •

Let $G = (V, E)$ be a distance regular graph of diameter d and order v, then for each $i \in \{0, \ldots, d\}$ we may define a $(0, 1)$-matrix A_i of dimension $v \times v$ as follows: rows and columns of A_i are indexed by the vertices of V and for every $x, y \in V$ the matrix entry $(A_i)_{xy}$ is defined by

$$(A_i)_{xy} = \begin{cases} 1 & \text{when } d(x, y) = i, \\ \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

Clearly A_1, also denoted as A, is the adjacency matrix of G and for every $i \in$

$\{0,\dots,d\}$ the *distance* matrices A_i satisfy the following four statements

$$\sum_{l=0}^{d} A_l = J, \quad A_0 = 1, \quad A_i = A_i^T, \quad AA_i = b_{i-1}A_{i-1} + a_iA_i + c_{i+1}A_{i+1} \tag{2.9}$$

where $A_{-1} = A_{d+1} = 0$ and b_{-1} and c_{d+1} are both undefined. The first three statements are straightforward. The fourth statement is valid as for every $x, y \in V$, the matrix entry $(AA_i)_{xy}$ is equal to either b_{i-1} when $d(x,y) = i-1$, a_i when $d(x,y) = i$ or c_{i+1} when $d(x,y) = i+1$.

Lemma 2.3.2. *Let $G = (V,E)$ be a distance regular graph of diameter d and order v, then there exist real polynomials $F_0 = 1$, $F_1, \dots, F_d$ with $\deg F_i = i$ such that $A_i = F_i(A)$. The coefficients of these polynomials can be expressed in terms of the intersection numbers a_i, b_i and c_i of G.*

Proof : From the second statement of (2.9) we find that $F_0(A) = 1$ and $F_1(A) = A$ are both polynomials in A with $\deg F_0 = 0$ and $\deg F_1 = 1$. Suppose that also for each $i \in \{2,\dots,l\}$ with $l < d$ we have that F_i is a polynomial in A with $\deg F_i = i$ and coefficients which can be expressed in term of the intersection numbers of G. From the fourth statement of (2.9) we find

$$F_{l+1}(A) = \frac{(A - a_i)F_l(A) - b_{i-1}F_{l-1}(A)}{c_{l+1}}, \tag{2.10}$$

a polynomial in A with $\deg F_{l+1} = l+1$. Again the coefficients can be expressed in terms of the intersection numbers of G. By induction we find that F_i is a polynomial in A with $\deg F_i = i$ for each $i \in \{0,\dots,d\}$. ■

By the fourth statement of (2.9) we can express A^k as $\sum_{l=0}^{d} q_l A_l$ for certain numbers q_l. Hence, by Lemma 2.3.2 we can express

$$A_iA_j = F_i(A)\,F_j(A) = \sum_{l=0}^{d} p_{ij}^l A_l \tag{2.11}$$

for certain numbers p_{ij}^l with $0 \le i, j, l \le d$.

From statements 1–3 of (2.9) and (2.11) if follows that the vertex set V of a distance regular graph of diameter d, together with an ordered set of relations

$\{R_0, \ldots, R_d\}$ where the relation R_i with $0 \le i \le d$ and $x, y \in V$, is defined by

$$xR_iy \iff d(x,y) = i,$$

forms a *d-class association scheme*. When we look at the matrix entries $(A_iA_j)_{x,y}$, we can observe that for each vertex $x, y \in V$ with $d(x,y) = l$, the number of vertices $z \in V$ with $d(x,z) = i$ and $d(y,z) = j$ is exactly the number p^l_{ij} for some $i, j, l \in \{0, \ldots, d\}$. The numbers p^l_{ij} are integers and satisfy the relations of Lemma 2.2.3. Moreover we have that

$$p^l_{ij} = 0 \text{ if } l > i + j \text{ or } l < |i - j|, \tag{2.12}$$

$$p^{i-1}_{1i} = b_{i-1}, \quad p^i_{1i} = a_i, \quad p^{i+1}_{1i} = c_{i+1}. \tag{2.13}$$

Consider finally all matrix entries $((AA_i)A_j)_{x,y}$ and $(A(A_iA_j))_{x,y}$ with $x, y \in V$ and $d(x,y) = l$, then from the fourth statement of (2.9) and (2.11) we find

$$b_{i-1}p^l_{i-1j} + a_ip^l_{ij} + c_{i+1}p^l_{i+1j} = p^{l-1}_{ij}c_l + p^l_{ij}a_l + p^{l+1}_{ij}b_l, \tag{2.14}$$

which can be used to compute the number p^l_{ij} recursively.

A special class of distance regular graphs are those of diameter 2. This class of distance regular graphs is equivalent to *strongly regular graphs*. For an overview of the theory of strongly regular graphs we refer to [14].

Definition 2.3.3. A connected graph $G = (V, E)$ of order v is *strongly regular* with parameters (v, k, λ, μ) if and only if it satisfies the following conditions:

- Each vertex $x \in V$ is adjacent to k vertices of V;
- For each pair of adjacent vertices $x, y \in V$ there are exactly λ vertices of V adjacent to both x and y;
- For each pair of non-adjacent vertices $x, y \in V$ there are exactly μ vertices of V adjacent to both x and y.

$\diamond$

The *complement* $\overline{G}$ of a strongly regular (v, k, λ, μ) graph G is again a strongly regular graph with parameters

$$(v,\ v - k - 1,\ v - 2k + \mu - 2,\ v - 2k + \lambda). \tag{2.15}$$

A graph of diameter 2 is distance regular if and only if it is *strongly regular* with $\mu > 0$. The intersection array of a strongly regular (v, k, λ, μ) graph G is given by the sequence

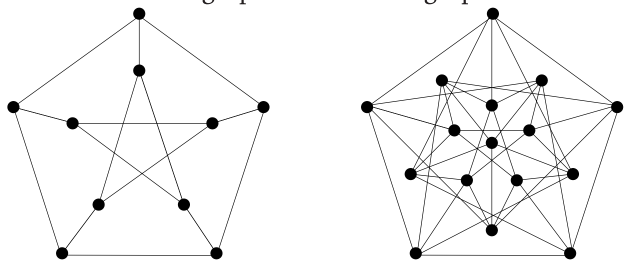
$$\{k,\, k-1-\lambda\,;\, 1,\, \mu\}. \tag{2.16}$$

The intersection matrices $L_0 = I$, L_1 and L_2 of a strongly regular (v, k, λ, μ) graph G can easily be expressed in terms of its parameters v, k, λ and μ:

$$L_1 = \begin{pmatrix} 0 & k & 0 \\ 1 & \lambda & k-\lambda-1 \\ 0 & \mu & k-\mu \end{pmatrix} \quad L_2 = \begin{pmatrix} 0 & 0 & v-k-1 \\ 0 & k-\lambda-1 & v-2k+\lambda \\ 1 & k-\mu & v-2k+\mu-2 \end{pmatrix} \tag{2.17}$$

If $\mu = k$ then we obtain trivial strongly regular graphs that are equivalent to complete multipartite graphs. A strongly regular graph is called non-trivial whenever $0 < \mu < k < v-1$.

Example 2.12. The Petersen graph and Clebsch graph are shown below.



Both graphs are strongly regular graphs with $(v, k, \lambda, \mu) = (10, 3, 0, 1)$ and $(16, 5, 0, 2)$, respectively. •

2.4 The Bose-Mesner algebra

First we recall some concepts from linear algebra. Let A be a real symmetric $v \times v$ matrix. Then all eigenvalues of A are real numbers. Denote the distinct eigenvalues of A by $\theta_0, \theta_1, \ldots$. The *minimal polynomial* M_A of A,

$$M_A(x) \overset{\text{def}}{=} (x-\theta_0)(x-\theta_1)\cdots$$

satisfies $M_A(A) = 0$. If F is any other polynomial with $F(A) = 0$, then F must be a multiple of M_A. In particular, $F = 0$ or $\deg F \geq \deg M_A$. The multiplicity

f_i of eigenvalue θ_i is the rank of the space generated by all eigenvectors of θ_i. The sum $f_0 + f_1 + \cdots$ of all multiplicities is equal to v. A is called *positive semidefinite* if and only if

$$x\, A\, x^T \geq 0 \tag{2.18}$$

for every row vector $x \in \mathbf{R}^{1\times v}$ and its transpose $x^T \in \mathbf{R}^{v\times 1}$.

Now, let $A_0, \ldots, A_d$ be as in (2.2). Consider the vector space

$$\mathcal{A} = \left\{ \sum_{i=0}^{d} c_i A_i : c_i \in \mathbf{R} \right\} \tag{2.19}$$

of all linear combinations of $A_0, \ldots, A_d$. Consider two elements of the vector space $\mathcal{A}$. By the fourth statement of (2.4) the product of any two elements again belongs to $\mathcal{A}$ and by Lemma 2.2.2 this product is commutative, hence the vector space $\mathcal{A}$ is a *commutative algebra*. This algebra $\mathcal{A}$ was first studied by Bose and Mesner [6] and is called the *Bose-Mesner algebra* of the association scheme. As the matrix entries of the elements $A_0, A_1, \ldots, A_d$ are either 0 or 1, we see from the first statement of (2.9) that the elements $A_0, A_1, \ldots, A_d$ are *linearly independent*, hence $\mathcal{A}$ has *dimension* $d + 1$.

The algebra $\mathcal{A}$ has many interesting properties which we will consider below. It is beyond the scope of this text to give proofs of all these properties in the *most general case* of association schemes. We shall therefore only prove them in the *restricted case* where the association scheme is a (connected) distance regular graph. For proofs of these properties in the general case, we refer to [1, 12].

Recall that we write $A = A_1$ for the adjacency matrix of a distance regular graph and $k = k_1$ for the degree of this graph, and that by Lemma 2.3.2 each $A_i = F_i(A)$ with $0 \leq i \leq d$ and $F_i(A)$ a polynomial in A with $\deg F_i = i$.

Lemma 2.4.1. *Let $G = (V, E)$ be a (connected) distance regular graph, then $\{1, A, A^2, \ldots, A^d\}$ is a basis for $\mathcal{A}$.*

Proof : From Lemma 2.3.2 we see that for each A_i with $i \in \{0, \ldots, d\}$, there exists a polynomial F_i with $\deg F_i = i$ such that $A_i = F_i(A)$. Hence the set $\{1, A, A^2, \ldots, A^d\}$ is a basis for $\mathcal{A}$. ■

Since $A^{d+1} \in \mathcal{A}$ and $\{1, A, A^2, \ldots, A^d\}$ is a basis for $\mathcal{A}$, we can express A^{d+1} in terms of a linear combination of $1, A, A^2, \ldots, A^d$. Let $A^{d+1} = \sum_{i=0}^{d} e_i A^i$ with

$e_i \in \mathbf{R}$. Consider the polynomial

$$Q(x) = x^{d+1} - \sum_{i=0}^{d} e_i x^i,$$

then $Q(A) = 0$, therefore the polynomial Q must be a multiple of the minimal polynomial M_A and thus clearly $Q = M_A$ as $\deg Q$ is minimal. Hence the minimal polynomial M_A of A must have degree $d+1$ and therefore A has $d+1$ distinct eigenvalues $\theta_0, \ldots, \theta_d$. It is customary to set $\theta_0 = k$, the degree of G, which is an eigenvalue of A with eigenvector the all-one vector [19]. The corresponding multiplicities shall be denoted by $f_0, \ldots, f_d$. (Later we shall prove that the multiplicity of the eigenvalue θ_0 is $f_0 = 1$.)

Let $i \in \{0, \ldots, d\}$, then we have for each eigenvector u of θ_i that $uA = \theta_i u$, hence for each $j \in \{1, \ldots, d\}$,

$$uA^j = (uA)A^{j-1} = \theta_i u A^{j-1} = \ldots = (\theta_i)^j u. \tag{2.20}$$

By Lemma 2.3.2 and (2.20) we find

$$uA_l = uF_l(A) = F_l(\theta_i)u \tag{2.21}$$

for each $i, l \in \{0, \ldots, d\}$. It follows that $F_l(\theta_i)$ is an eigenvalue of A_l with the same eigenvectors u of θ_i and hence also the same multiplicity f_i.

The $(d+1) \times (d+1)$ matrix P with entries

$$P_{ij} = F_j(\theta_i) \tag{2.22}$$

is called the *eigenmatrix* of the scheme. By the above, P_{ij} is an eigenvalue of A_j. (Later we shall prove that the eigenmatrix P can be computed from the intersection numbers.)

Lemma 2.4.2. *Define*

$$E_i = \prod_{k=0, k \neq i}^{d} \frac{A - \theta_k}{\theta_i - \theta_k}. \tag{2.23}$$

Then we have

$$E_i A = \theta_i E_i \tag{2.24}$$

$$E_i E_j = \begin{cases} E_i, & \text{when } i = j, \\ 0, & \text{when } i \neq j. \end{cases} \tag{2.25}$$

The matrix $E_i^2 = E_i$ *has eigenvalue* 1 *with multiplicity* f_i *and eigenvalue* 0 *with multiplicity* $v - f_i$. *Hence* E_i *has rank* f_i.

Proof : For every $i \in \{0, \dots, d\}$ we have

$$E_i\,(A - \theta_i) = \left(\prod_{k=0,k\neq i}^{d} \frac{1}{\theta_i - \theta_k} \right) M_A(A) = 0.$$

Hence we find that $E_i A = \theta_i E_i$.

By (2.24) we have for every $i, j \in \{0, \dots, d\}$

$$E_i E_j = E_i \left(\prod_{k=0,k\neq j}^{d} \frac{A - \theta_k}{\theta_j - \theta_k} \right) = E_i \left(\prod_{k=0,k\neq j}^{d} \frac{\theta_i - \theta_k}{\theta_j - \theta_k} \right)$$

and hence $E_i E_j = E_i$ when $i = j$ and $E_i E_j = 0$ when $i \neq j$.

For every $i \in \{0, \dots, d\}$ define the polynomial

$$H_i(x) = \prod_{k=0,k\neq i}^{d} \frac{x - \theta_k}{\theta_i - \theta_k},$$

then $E_i = H_i(A)$. Every eigenvector u for an eigenvalues θ_j of A, is an eigenvector for the eigenvalue $H_i(\theta_j)$ of E_i. Because $H_i(\theta_i) = 1$ and $H_i(\theta_j) = 0$ when $i \neq j$, it follows that E_i has eigenvalue 1 with multiplicity f_i, and eigenvalue 0 with multiplicity

$$\sum_{k=0,k\neq i}^{d} f_k = v - f_i.$$

■

E_i is called the *minimal idempotent* associated with eigenvalue θ_i.

Lemma 2.4.3. *For every* $i \in \{0, \dots, d\}$ *we have* E_i *is positive semidefinite.*

Proof : By Lemma 2.4.2 we find that $E_i^2 = E_i$. Hence for every row vector

$x \in \mathbf{R}^{1\times v}$ and its transpose $x^T \in \mathbf{R}^{v\times 1}$, we have

$$\begin{aligned} x\,E_i\,x^T &= x\,E_i{}^2\,x^T \\ &= (x\,E_i)(E_i{}^T\,x^T) \\ &= (x\,E_i)(x\,E_i)^T \\ &= y_1{}^2 + \ldots + y_v{}^2 \geq 0 \end{aligned}$$

with $y = x\,E_i$. ■

From (2.25) we find for each $j \in \{0, \ldots, d\}$ that

$$\left(\sum_{i=0}^{d} c_i E_i\right) E_j = c_j E_j$$

with $c_i \in \mathbf{R}$. Since E_j has eigenvalue 1 with multiplicity $f_j \neq 0$, we find that $E_j \neq 0$. Hence $\sum_{i=0}^{d} c_i E_i = 0$ if and only if $c_j = 0$ for all $j \in \{0, \ldots, d\}$. Therefore all minimal idempotents E_i are linearly independent. Consequently the set of minimal idempotents $\{E_0, \ldots, E_d\}$ forms a basis of the $d+1$ dimensional algebra $\mathcal{A}$.

The following lemma shows how the matrices A_i can be expressed in terms of the minimal idempotents $E_0, \ldots, E_d$.

Lemma 2.4.4. *For every $j \in \{0, \ldots, d\}$ we have*

$$A_j = \sum_{i=0}^{d} P_{ij} E_i.$$

Proof : From (2.24) we have that $E_i A = \theta_i E_i$ for each $i \in \{0, \ldots, d\}$. Hence

$$E_i A^j = (\theta_i)^j E_i \tag{2.26}$$

with $i, j \in \{0, \ldots, d\}$. Using Lemma 2.3.2 and by (2.22) and (2.26) we find that

$$E_i A_j = E_i F_j(A) = F_j(\theta_i) E_i = P_{ij} E_i \tag{2.27}$$

for each $i, j \in \{0, \ldots, d\}$.

Since the minimal idempotents $E_0, \dots, E_d$ form a basis for $\mathcal{A}$, we have that

$$A_i = \sum_{j=0}^{d} a_{ij} E_j.$$

for each $i \in \{0, \dots, d\}$ and for some $a_{ij} \in \mathbf{R}$. Let $i, j \in \{0, \dots, d\}$ then by (2.25) the product

$$E_j A_i = E_j \sum_{l=0}^{d} a_{il} E_l = a_{ij} E_j, \tag{2.28}$$

hence by (2.27) we find that $a_{ij} = P_{ji}$. ■

We may also express each minimal idempotent E_i in terms of $A_0, \dots, A_d$ in a unique way. Using this property we may define the unique coefficients Q_{ji} as follows :

$$v E_i = \sum_{j=0}^{d} Q_{ji} A_j. \tag{2.29}$$

(Note the extra factor v in this expression). The $(d+1) \times (d+1)$ matrix Q with matrix entries $(Q)_{ij} = Q_{ij}$ is called he *dual eigenmatrix* of the scheme.

From Lemma 2.4.4 and (2.29), we find that

$$v E_i = \sum_{j=0}^{d} Q_{ji} \sum_{l=0}^{d} P_{lj} E_l = \sum_{l=0}^{d} (PQ)_{li} E_l$$

$$v A_i = \sum_{j=0}^{d} P_{ji} \sum_{l=0}^{d} Q_{lj} A_l = \sum_{l=0}^{d} (QP)_{li} A_l$$

for every $i \in \{0, \dots, d\}$. Hence the eigenmatrix P and the dual eigenmatrix Q satisfy

$$PQ = QP = v. \tag{2.30}$$

We shall now prove that the eigenmatrix P, the dual eigenmatrix Q and the multiplicities f_i can be computed from the intersection parameters.

Let B be an element of $\mathcal{A}$. Consider the linear operator $\rho(B)$ on $\mathcal{A}$ which maps $X \in \mathcal{A}$ onto $\rho(X) \cdot B \overset{\text{def}}{=} BX \in \mathcal{A}$.

Since $A_0, \ldots, A_d$ is a basis for $\mathcal{A}$, for every $B \in \mathcal{A}$ we have that $B = \sum_{i=0}^{d} b_i A_i$ is a linear combination of $A_0, \ldots, A_d$ with $b_i \in \mathbf{R}$. We may represent every $B \in \mathcal{A}$ as a *column vector* $(b_0, \ldots, b_d)^T$. For example: the identity matrix 1 is represented by $(1, 0, \ldots, 0)^T$, A is represented by $(0, 1, 0, \ldots, 0)^T$ and J, the all–1 matrix by $(1, 1, \ldots, 1)^T$. With this representation, every linear operator on $\mathcal{A}$ corresponds to a $(d+1) \times (d+1)$ matrix which acts on the left.

Let $i \in \{0, \ldots, d\}$ and $B \in \mathcal{A}$ then using (2.1) we find,

$$\begin{aligned}
\rho(A_i) \cdot B &= \rho(A_i) \cdot \sum_{j=0}^{d} b_j A_j = \sum_{j=0}^{d} b_j \, \rho(A_i) \cdot A_j \\
&= \sum_{j=0}^{d} b_j \sum_{l=0}^{d} p_{ij}^{l} A_l = \sum_{j=0}^{d} \sum_{l=0}^{d} b_j \, p_{ij}^{l} A_l \\
&= \sum_{l=0}^{d} \sum_{j=0}^{d} (L_i)_{lj} \, b_j \, A_l .
\end{aligned}$$

Then the linear operator $\rho(A_i)$ corresponds to

$$\rho(A_i) \leftrightarrow L_i = \begin{pmatrix} p_{i0}^{0} & \cdots & p_{id}^{0} \\ \vdots & \ddots & \vdots \\ p_{i0}^{d} & \cdots & p_{id}^{d} \end{pmatrix},$$

that is, the $(d+1) \times (d+1)$ intersection matrix.

In the following Lemma we give the connection between the eigenvalues of the matrices A_i and the eigenvalues of the intersection matrices L_i.

Lemma 2.4.5. *Let $i \in \{0, \ldots, d\}$, then the eigenvalues of the intersection matrix L_i are the eigenvalues of A_i but with multiplicities 1. The eigenvectors of L_i are the vectors $(Q_{0j}, \ldots, Q_{dj})^T$, with $j \in \{0, \ldots, d\}$.*

Proof : Let $i, j \in \{0, \ldots, d\}$ then by (2.27) and (2.29) we have

$$\rho(A_i) \cdot \sum_{l=0}^{d} Q_{lj} A_l = (\sum_{l=0}^{d} Q_{lj} A_l) A_i = v \, E_j \, A_i = v \, P_{ji} \, E_j = P_{ji} \sum_{l=0}^{d} Q_{lj} \, A_l ,$$

or in coordinate vector representation

$$\rho(A_i)\begin{pmatrix} Q_{0j} \\ \vdots \\ Q_{dj} \end{pmatrix} = L_i \begin{pmatrix} Q_{0j} \\ \vdots \\ Q_{dj} \end{pmatrix} = P_{ji} \begin{pmatrix} Q_{0j} \\ \vdots \\ Q_{dj} \end{pmatrix}.$$

Hence P_{ji} is an eigenvalue of the intersection matrix L_i with corresponding eigenvector $(Q_{0j}, \ldots, Q_{dj})$. Since all eigenvalues of A_i are distinct, we find that each of the eigenvalues $P_{0i}, \ldots, P_{di}$ must have multiplicity 1. ■

In the following Lemma we show how to compute the multiplicities $f_0, \ldots, f_d$ of the eigenvalues $\theta_0, \ldots, \theta_d$ of A.

Lemma 2.4.6. *The minimal idempotent associated with eigenvalue θ_0 is $E_0 = \frac{1}{v}J$ and for each $i \in \{0, \ldots, d\}$ the multiplicity of the eigenvalue θ_i of A is $f_i = Q_{0i}$. Moreover $Q_{j0} = 1$ for each $j \in \{0, \ldots, d\}$ and in particular $f_0 = 1$.*

Proof : Let $i \in \{0, \ldots, d\}$. Then we have

$$\sum_{j=0}^{d} P_{ji} f_j = \mathrm{Tr}(A_i) = \begin{cases} v & \text{whenever } i = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Let $l \in \{0, \ldots, d\}$. Then we have

$$\sum_{i=0}^{d} \sum_{j=0}^{d} Q_{il} P_{ji} f_j = \sum_{i=0}^{d} Q_{il} \,\mathrm{Tr}(A_i) = v\, Q_{0l}$$

and

$$\sum_{i=0}^{d} \sum_{j=0}^{d} Q_{il} P_{ji} f_j = \sum_{j=0}^{d} (PQ)_{lj} f_j = v\, f_l.$$

since $PQ = QP = v$. Hence $f_l = Q_{0l}$. Also we have

$$\rho(A_i) \cdot J = J\, A_i = k_i\, J.$$

and by (2.27)

$$\rho(A_1) \cdot E_0 = P_{01}\, E_0 = k_1\, E_0$$

since $\theta_0 = P_{01} = k_1$. Hence both E_0 and J are eigenvectors of $\rho(A_1)$ with the same eigenvalue k_1 of multiplicity 1. Therefore $E_0 = cJ$ with $c \in \mathbf{R} \setminus \{0\}$. By (2.25) we have $(E_0)^2 = E_0$ and therefore $E_0 = \frac{1}{v}J$. Hence $A_iE_0 = \frac{k_i}{v}J = \frac{k_i}{v}\sum_{l=0}^{d} Q_{l0}\,A_l$, and therefore $Q_{l0} = 1$. ■

Example 2.13. As a first example we compute the values of the eigenmatrix P and the dual eigenmatrix Q for the Perkel graph G on 57 vertices. The Perkel graph is the unique (up to isomorphism) distance regular graph of degree 6 and diameter 3 with intersection array $\{6,5,2;1,1,3\}$. Using (2.12–2.14) we can compute the corresponding intersection matrices

$$L_1 = \begin{pmatrix} 0 & 6 & 0 & 0 \\ 1 & 0 & 5 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 3 & 3 \end{pmatrix} \quad L_2 = \begin{pmatrix} 0 & 0 & 30 & 0 \\ 0 & 5 & 15 & 10 \\ 1 & 3 & 14 & 12 \\ 0 & 3 & 18 & 9 \end{pmatrix} \quad L_3 = \begin{pmatrix} 0 & 0 & 0 & 20 \\ 0 & 0 & 10 & 10 \\ 0 & 2 & 12 & 6 \\ 1 & 3 & 9 & 7 \end{pmatrix}$$

in a recursive manner. Hence using Lemma 2.4.5 we find that

$$P = \begin{pmatrix} 1 & 6 & 30 & 20 \\ 1 & \frac{3+\sqrt{5}}{2} & \frac{-5+3\sqrt{5}}{2} & -2\sqrt{5} \\ 1 & \frac{3-\sqrt{5}}{2} & \frac{-5-3\sqrt{5}}{2} & 2\sqrt{5} \\ 1 & -3 & 3 & -1 \end{pmatrix}$$

By (2.30) we have $Q = 57\,P^{-1}$. Hence we find that

$$Q = \begin{pmatrix} 1 & 18 & 18 & 20 \\ 1 & \frac{9}{2} + \frac{3\sqrt{5}}{2} & \frac{9}{2} - \frac{3\sqrt{5}}{2} & -10 \\ 1 & -\frac{3}{2} + \frac{9\sqrt{5}}{10} & -\frac{3}{2} - \frac{9\sqrt{5}}{10} & 2 \\ 1 & -9\sqrt{5} & 9\sqrt{5} & -1 \end{pmatrix}$$

For every $i \in \{0,1,2,3\}$ the matrix entry P_{1i} is an eigenvalue of the adjacency matrix A with multiplicity Q_{0i}. Hence the graph spectrum of the Perkel graph is given by

$$\operatorname{spec}(G) = \left\{ 6^1, \left(\frac{3+\sqrt{5}}{2}\right)^{18}, \left(\frac{3-\sqrt{5}}{2}\right)^{18}, (-3)^{20} \right\}$$

●

Example 2.14. As a second example we consider the general case of strongly regular graphs. Let $G = (V, E)$ be a non-trivial strongly regular graph with parameter set (v, k, λ, μ). The intersection matrices are given by (2.17), which allow us to compute the eigenmatrix P and the dual eigenmatrix Q for a strongly regular (v, k, λ, μ) graph G. Using Lemma 2.4.5 we find that

$$P = \begin{pmatrix} 1 & k & v-k-1 \\ 1 & r & -r-1 \\ 1 & s & -s-1 \end{pmatrix} \tag{2.31}$$

where r and s can be found as solutions of

$$\theta^2 + (\mu - \lambda)\,\theta + \mu - k = 0 \tag{2.32}$$

with $r \geq 0$ and $s \leq -1$. By (2.30) we have $Q = v\,P^{-1}$. Hence we find that

$$Q = \begin{pmatrix} 1 & f & g \\ 1 & \frac{fr}{k} & \frac{gs}{k} \\ 1 & -f\frac{r+1}{v-k-1} & -g\frac{s+1}{v-k-1} \end{pmatrix} \tag{2.33}$$

where f and g can be found as solutions of

$$\begin{aligned} 1 + f + g &= v \\ k + fr + gs &= 0 \end{aligned} \tag{2.34}$$

For every $i \in \{0, 1, 2\}$ the matrix entry P_{1i} is an eigenvalue of the adjacency matrix A with multiplicity Q_{0i}. Hence the graph spectrum of a strongly regular (v, k, λ, μ) graph G is given by

$$\text{spec}(G) = \{k^1, r^f, s^g\} \tag{2.35}$$

•

Example 2.15. As a last example we show how the eigenmatrix P and the dual eigenmatrix Q can be computed in the general case of an association scheme. The example we have chosen is not typical, but illustrates very well the differences between the general case and the more specific case of the distance regular graphs. We consider the intersection numbers of a cyclotomic three class association scheme of order 16, i.e.,

$$L_1 = \begin{pmatrix} 0 & 5 & 0 & 0 \\ 1 & 0 & 2 & 2 \\ 0 & 2 & 2 & 1 \\ 0 & 2 & 1 & 2 \end{pmatrix} \quad L_2 = \begin{pmatrix} 0 & 0 & 5 & 0 \\ 0 & 2 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 0 & 1 & 2 & 2 \end{pmatrix} \quad L_3 = \begin{pmatrix} 0 & 0 & 0 & 5 \\ 0 & 2 & 1 & 2 \\ 0 & 1 & 2 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix}$$

Also in this general case we are allowed to use Lemma 2.4.5 to compute te eigenmatrix P. We find

$$P = \begin{pmatrix} 1 & 5 & 5 & 5 \\ 1 & -3 & 1 & 1 \\ 1 & 1 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{pmatrix}$$

By (2.30) we have $Q = 16\, P^{-1}$. It turns out that the dual eigenmatrix Q is equal to P. What makes this case different from the distance regular case, is that each of the matrices A_i has only 3 distinct eigenvalues, while the number of relations of Ω, and hence the dimension of the Bose-Mesner algebra, is 4. •

2.5 Feasibility criteria

In what follows we will only consider non-trivial cases of d-class association schemes and in particular strongly regular and distance regular graphs.

A list of feasible intersection matrices for 3-class association schemes on at most 100 vertices together with spectra of their relations, known constructions and known nonexistence results can be obtained from [101]. In the case of distance regular graphs, the following proposition gives some restrictions on putative intersection arrays.

Proposition 2.5.1 ([12]). *For any distance regular $G = (V, E)$ of diameter d the intersection array must satisfy the following restrictions.*

- $k = b_0 > b_1 \geq b_2 \geq \ldots \geq b_{d-1} > b_d = 0$ *and* $1 = c_1 \leq c_2 \leq \ldots \leq c_d \leq k$.
- $c_i \leq b_j$ *whenever* $i + j \leq d$
- *for every* $i, j, l \in \{0, \ldots, d\}$ *the numbers* p_{ij}^l *and* k_i *are nonnegative integers*
- *the multiplicities of the eigenvalues of* A *must be integers.*

Proof : For any x, $y \in V$ at distance d, let $x = \gamma_0, \gamma_1, \ldots, \gamma_d = y$ be a path of length d. Counting the number of neighbours of γ_i at distance $i - 2$ from γ_1 yields $c_i \geq c_{i-1}$, and counting neighbours of γ_i at distance i from γ_1 yields

$b_i \leq b_{i-1}$. Hence the first statement is valid. The second statement is also valid as counting the neighbours of γ_i at distance $i-1$ from γ_0 yields $c_i \leq b_j$. Finally, statements 3–4 are straightforward. ∎

We shall call an intersection array feasible whenever it satisfies each of these restrictions. A list of all feasible intersection arrays together with graph spectra, known constructions and known nonexistence results for primitive distance regular graphs with diameter 3 on at most 1024 vertices, for non-bipartite distance regular graphs with diameter 4 on at most 4096 vertices, and arbitrary distance-regular graphs of diameter at least 5 on at most 4096 vertices can be obtained from [12].

For strongly regular graphs, we have the following proposition.

Proposition 2.5.2. *For any non-trivial strongly regular (v, k, λ, μ) graph G the parameters v, k, λ, μ and the multiplicities of the eigenvalues of the adjacency matrix A must be nonnegative integers. Moreover the parameters v, k, λ, μ must satisfy the following restrictions:*

- $0 < \mu < k < v - 1$
- $\mu(v - k - 1) = k(k - 1 - \lambda)$
- $v - 2k + \mu - 2 \geq 0$ *and* $v - 2k + \lambda \geq 0$

Proof : The first restrictions are obvious. Choose a vertex x of G. Counting in two ways the number of edges between the set of vertices adjacent to x and the set of vertices non-adjacent to x yields the restriction $\mu(v - k - 1) = k(k - 1 - \lambda)$. Restrictions $v - 2k + \mu - 2 \geq 0$ and $v - 2k + \lambda \geq 0$ stem from (2.17). ∎

We shall call a (v, k, λ, μ) parameter set *feasible* whenever it satisfies all these restrictions. If (v, k, λ, μ) is feasible, then either the eigenvalues r and s are integral, or there is a t such that $v = 4t + 1$, $k = 2t$, $\lambda = t - 1$, $\mu = t$ and $r, s = \frac{(-1 \pm \sqrt{v})}{2}$, the so-called *half case*. All feasible (v, k, λ, μ) parameter sets for strongly regular graphs on at most 280 vertices together with graph spectra, known constructions and known nonexistence results can be obtained from [14].

3 Generation algorithms.

”Space is big. You just won’t believe how vastly, hugely, mind-bogglingly big it is.”[D. ADAMS, THE HITCHHIKER’S GUIDE TO THE GALAXY]

Combinatorial classification deals with the problem of exhaustively generating, up to isomorphism, all combinatorial objects satisfying a set of constraints. Quite often in an algorithmic context, the more common term *isomorph-free exhaustive generation* is used instead. Generation algorithms consist of two major components. A first component deals with exhaustively generating at least one representative from every isomorphism class, while the second component is responsible for removing isomorphic objects from consideration so as to obtain isomorph-free generation.

In this chapter we describe some of the general techniques which are commonly used in the design of such generation algorithms. Parts of this chapter are based on [17, 29, 65, 67, 70, 94]. In Section 3.1 we introduce the concepts of constraint networks and backtracking which allow us to describe exhaustive generation in a general framework, while in Sections 3.2 and 3.3 we discuss isomorphism and isomorphism rejection techniques in a group-theoretic framework. These general techniques will serve as a basis for the more customized generation algorithms used in the classification of d-class association schemes, and in particular strongly regular and distance regular graphs. These more specific algorithms shall be discussed throughout Chapters 4, 5 and 6.

3.1 Exhaustive generation

Combinatorial objects are finite and discrete structures which makes corresponding classification problems susceptible to be tackled with exhaustive search methods. Exhaustive search methods systematically build or investigate all possible states within some search space. In the context of combinatorial classification, exhaustive generation is typically exemplified by *backtracking* or *backtrack search* [49] — probably the most faimiliar exhaustive search technique. In order to outline the backtracking paradigm in the context of combinatorial classification, we first introduce some concepts and notations. Generally, combinatorial classification can be regarded as finding all solutions of a *constraint network* [29].

Definition 3.1.1. A *constraint network* $\mathcal{R} = (X, D, C)$ consists of a finite set of *variables* $X = \{x_1, \ldots, x_n\}$ with respective set of non-empty *domains* $D = \{D_1, \ldots, D_n\}$ and a set of *constraints* $C = \{C_1, \ldots, C_t\}$. $\diamond$

A domain D_i lists all possible values for its corresponding variable x_i. A constraint C_j involves a certain subset of variables and specifies the allowable combinations of values for that particular subset. When a variable is assigned a value from its domain, we say that the variable is instantiated, otherwise we say that the variable is uninstantiated. For notational simplicity we assign an uninstantiated variable the undefined value $'?'$. An instantiation of a subset of variables is an assignment from its domain to each variable in that subset. More precisely, an instantiation of a set of variables $\{x_{i1}, \ldots, x_{ik}\}$ is a tuple of ordered pairs $(\langle x_{i1}, d_{i1}\rangle, \ldots, \langle x_{ik}, d_{ik}\rangle)$, where each ordered pair $\langle x, d\rangle$ denotes an assignment of the value d to the variable x, where d is in the domain of x. We shall abbreviate such instantiation to $(d_{i1}, \ldots, d_{ik})$ whenever the subset of variables is clear from context. A *solution* of a constraint network is a complete instantiation of all of its variables such that all of its constraints are satisfied. A partial instantiation of a constraint network is *feasible* if it satisfies all of its constraints which have no uninstantiated variables. If a partial instantiation, say $(d_1, \ldots, d_k)$ is feasible, then all partial instantiations $(d_1, \ldots, d_j)$ with $j < k$ are also feasible. Hence if a partial instantiation $(d_1, \ldots, d_k)$ is not feasible, then also all extensions $(d_1, \ldots, d_l)$ with $l > k$ are not feasible, or to put it differently, no extension of the partial instantiation $(d_1, \ldots, d_k)$ could ever lead to a solution of the constraint network.

Example 3.1. Consider the exhaustive generation of the set Δ of all regular graphs of order 4 and degree 2. In terms of a constraint network, such graphs are solutions of the constraint network $\mathcal{R} = (X, D, C)$ with variables $X = \{x_1, x_2, \ldots, x_6\}$ and corresponding domains $D = \{D_1, D_2, \ldots, D_6\}$ such that each domain $D_i = \{0, 1\}$.

$$A = \begin{pmatrix} 0 & x_1 & x_2 & x_4 \\ x_1 & 0 & x_3 & x_5 \\ x_2 & x_3 & 0 & x_6 \\ x_4 & x_5 & x_6 & 0 \end{pmatrix}$$

The set of constraints C can easily be expressed using the matrix representation A above. An instantiation, partial or complete, is feasible if each row and column of A contains at most 2 ones and at most 2 zeros. Given a solution of $\mathcal{R}$, the resulting matrix A — with each variable instantiated — corresponds to the adjacency matrix of a regular graph of order 4 and degree 2. •

Backtracking is a recursive algorithm in which at each time a partial instantiation of the set of variables $X = \{x_1, \ldots, x_n\}$ of the constraint network $\mathcal{R} = (X, D, C)$ is maintained. The algorithm starts with all n variables uninstantiated. At each step in the recursion, one particular variable is chosen, and systematically all values from its domain are assigned to it in turn. For each such value, the feasibility of the partial instantiation is checked, that is, the consistency of the partial instantiation with all constraints which have no uninstantiated variables is evaluated. In case of feasibility, a recursive call is invoked. Otherwise, no recursive call is invoked since no extension of the partial instantiation could ever lead to a solution of the constraint network. If all values turn out to be infeasible, then a *dead-end* occurs. When all values of its domain have been tried in turn, the algorithm backtracks to the invoking procedure. A pseudo code description of a general backtracking algorithm is given in Algorithm 3.1. As control strategy, at each step in the recursion the uninstantiated variable with the smallest index is selected.

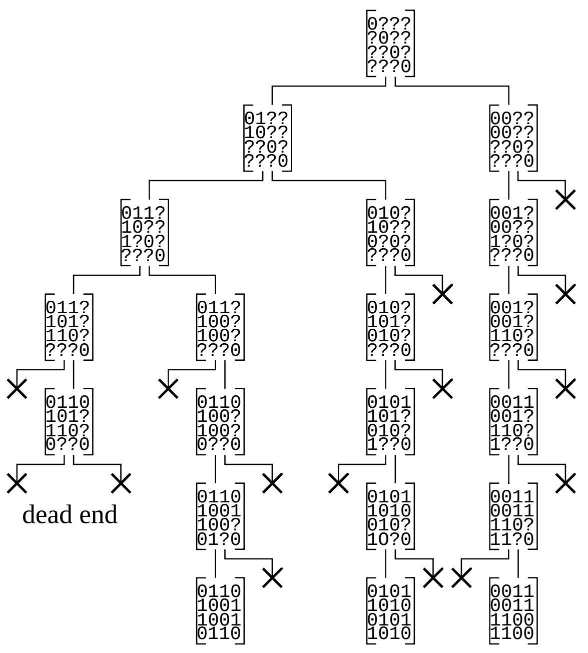
We can easily visualize backtrack search as a *depth-first* traversal of a *search*

Algorithm 3.1 General backtrack algorithm for a constraint network $\mathcal{R} = (X, D, C)$ with n variables.

```
procedure search( )
1: backtrack(( ),0)

procedure backtrack(($d_1, \ldots, d_k$): instantiation, $k$: int))
1: if $k = n$ then
2:    report ($d_1, \ldots, d_n$) as a solution
3: else
4:    for all $d_{k+1} \in D_{k+1}$ do
5:       if ($d_1, \ldots, d_{k+1}$) is feasible then
6:          backtrack(($d_1, \ldots, d_{k+1}$),k+1)
```

tree or a so-called *recursion tree*. Nodes of this search tree correspond to partial feasible instantiations, while branches represent the systematic extension and collapsing from which one partial feasible instantiation can be obtained from another. In the forward phase of the backtrack algorithm, the extension of a partial feasible instantiation corresponds to a traversal of the search tree to a deeper level, while in the backward phase of the backtracking algorithm the collapsing is represented by a traversal in the opposite direction. If a partial instantiation is not feasible, then the entire subtree rooted at the corresponding node can be *pruned*. A *leaf* at the deepest level n in the recursion tree corresponds to a solution of the constraint network $\mathcal{R} = (X, D, C)$. The set of all nodes in the recursion tree shall be denoted by $N_{\mathcal{R}}$.

Example 3.2. For the constraint network $\mathcal{R} = (X, D, C)$ as defined in Example 3.1, the corresponding search tree is shown in Figure 3.1. The control strategy applied during backtrack search is to select at each step in the recursion, the uninstantiated variable x_i with smallest index i. At the root of the tree, all variables are uninstantiated. Left children correspond to the extension of a partial feasible instantiation by assigning the current variable the value 1 from its domain, while right children correspond to the extension of a partial feasible instantiation by assigning the current variable the value 0 from its domain. A cross means that the extension leads to a partial instantiation which is no longer feasible and therefore the subtree rooted at that particular node can be pruned. •



Figuur 3.1: Search tree corresponding to the exhaustive generation of all regular graphs of order 4 and degree 2.

Constraint networks and backtracking merely provide us with a general framework which we can use to solve combinatorial classification problems. When designing such exhaustive generation algorithms for concrete combinatorial classifications problems, a translation of these problems into this framework is still required. Mostly, an adequate translation depends on how we can exploit the mathematical properties of the combinatorial objects at hand effectively during traversal so as to reduce the size of the corresponding search tree, which of itself exhibits an exponential behaviour. A series of basic principles for designing fast backtracking algorithms occurs in [74].

When designing generation algorithm, it is good to keep in mind that a general backtracking algorithm suffers from *trashing* [29, 67], that is, repeatedly rediscovering the same partial inconsistencies and the same partial successes during search. The performance of general backtracking algorithms can be improved by dynamically adapting the algorithm's control strategy during search. These methods for dynamically improving the performance of general backtrack search, can be divided into two classes corresponding with the algorithm's forward and backward phase.

During the forward phase of the bactracking algorithm typically two different types of methods are invoked when the algorithm is about to select a new variable or to assign a new value to the currently selected variable [58, 67].

looking ahead When the algorithm assigns a value to the currently selected variable, values from the domains of all still uninstantiated variables which conflict with the current partial instantiation can be removed. We denote by D'_i, the subset of the domain D_i which has it conflicting values removed. When selecting x_i as the current variable, only the still remaining values from D'_i have to be assigned to x_i in turn. When a variable $d \in D_i$ has been removed when instantiating a previous variable x_c, then the value d has to be restored when that variable is reassigned a new value. Note that not necessarily all uninstantiated variables and all constraints have to be evaluated.

dynamic variable ordering When the algorithm is about to select a new variable, its control strategy can dynamically decide during search which variable to select next. A frequently used heuristic is to choose the variable which is most constrained, that is, the variable with the least number of values remaining in its domain. If the domain of some uninstantiated variable is empty, then that variable is selected as the new variable and a dead-end occurs immediately. Applying this dynamic variable ordering strategy implies that the order of variable instantiation does not need to be the same in different branches of the recursion tree.

Look-ahead strategies involve an extra cost after each variable instantiation. However, dead-ends occur sooner during search, and mostly a smaller portion of the search space needs to be traversed.

For the backward phase there are two types of techniques which are typically used when preparing to backtrack after encountering a dead-end.

looking back When the algorithm encounters a dead-end, then normally the current partial instantiation is collapsed one step backwards. Identification of the reasons for this particular dead-end often allows us to avoid irrelevant backtrack points. The backtrack algorithm is modified so that it goes back immediately to the origin of failure — instead of just to the preceding variable.

constraint recording When the algorithm encounters a dead-end, the reasons for this dead-end are recorded in the form of new constraints. This is done to avert that the same conflicts reappear later in the search.

3.2 Isomorphism in a group-theoretic framework

Any profound discussion of the isomorphism relation associated with most combinatorial objects and — from a more algorithmic point of view — of isomorphism rejection techniques should be treated within a group-theoretic framework.

3.2.1 Group theory

First we recall some concepts from group theory [17, 94].

Definition 3.2.1. A *group* $(G, *)$ is a non-empty set G with a binary operator $*$, such that:

- $\pi_1 * \pi_2 \in G$ for all $\pi_1, \pi_2 \in G$ *(closure)*
- $(\pi_1 * \pi_2) * \pi_3 = \pi_1 * (\pi_2 * \pi_3)$ for all $\pi_1, \pi_2, \pi_3 \in G$ *(associativity)*
- there exists id $\in G$ such that $\pi * \mathrm{id} = \mathrm{id} * \pi$ for all $\pi \in G$ *(identity element)*
- for all $\pi \in G$, there exists $\pi^{-1} \in G$ such that $\pi * \pi^{-1} = \pi^{-1} * \pi = \mathrm{id}$ *(inverse element)*

◇

If the binary operator $*$ is clear from context, it is customary to simply state that G is a group. Also we often write $\pi_1\,\pi_2$ instead of $\pi_1 * \pi_2$. A group G is *finite* if G is a finite set. We will only consider finite groups. The size $|G|$ of the set G is called the *order* of the group G. When G is finite, for each element $\pi \in G$ there is always a smallest positive integer m such that

$$\pi^m = \underbrace{\pi * \pi * \ldots * \pi}_{m\ times} = \mathrm{id}$$

The integer m is called the *order* of the element π. Let S be a subset of G. The set S *generates* G if each element $\pi \in G$ can be written as $\pi = \sigma_1 * \sigma_2 * \ldots * \sigma_m$ with $\sigma_1, \sigma_2, \ldots, \sigma_m \in S$ for some m only depending on π. We call S a set of *generators* for G and denoted this by $G = \langle S \rangle$.

A *permutation* is a bijection of a non-empty set X onto itself. Let π be permutation of X, then x^π denotes the image of $x \in X$ under π. The *product* $\pi\,\sigma$ of two permutations π and σ of X satisfies $x^{\pi\,\sigma} = (x^\pi)^\sigma$.

Definition 3.2.2. The *symmetric group* on a non-empty set X, is the group whose underlying set is the set of all permutation from X onto itself and whose binary operation is the multiplication of permutations. ◇

The *symmetric group* on X shall be denoted by $\mathrm{Sym}(X)$ or by $\mathrm{Sym}(n)$ whenever $X = \{1, \ldots, n\}$. Note that $|\mathrm{Sym}(n)| = n!$. The *degree* of a permutation group on X is $|X|$.

Definition 3.2.3. A *permutation group* is a group G whose elements are permutations of a given set X and whose group operator is the multiplication of permutations in G. ◇

Let $x \in X$ and $\pi \in \text{Sym}(X)$, then π *stabilizes* x if $x^\pi = x$; otherwise π *moves* x. Two permutations π, $\sigma \in \text{Sym}(X)$ are called *disjoint* if and only if for every element of $x \in X$ either π stabilizes x and σ moves x or vice versa. Let $x_1, x_2, \ldots, x_r \in \{1, \ldots, n\}$, $r \geq 2$ be distinct. The permutation $\pi \in \text{Sym}(n)$ with images $x_1^\pi = x_2$, $x_2^\pi = x_3$, …, $x_{r-1}^\pi = x_r$, $x_r^\pi = x_1$, and which stabilizes all remaining integers, is called an *r-cycle* and is denoted by $(x_1\, x_2\, \ldots\, x_r)$. Every permutation $\pi \in \text{Sym}(n)$ is either the identity or product of one or more pairwise disjoint cycles.

Example 3.3. The permutation $\pi \in \text{Sym}(6)$ with $1^\pi = 2, 2^\pi = 4, 3^\pi = 6, 4^\pi = 1, 5^\pi = 5$ and $6^\pi = 3$ can be written as a product of cycles $\pi = (1\,2\,4)\,(3\,6)$. •

Definition 3.2.4. Let $(G, *)$ be a group. Let $H \subset G$. Then $(H, *)$ is called a *subgroup* of $(G, *)$ when $(H, *)$ is itself a group. ◇

When the binary operator $*$ is clear from context, we shall say that H is a subgroup of G and write $H \leq G$ to indicate this fact. If $H \neq G$ then H is said to be a *proper subgroup* of G, and we write $H < G$.

Example 3.4. The 5-cycle $(1\,2\,3\,4\,5)$ generates a subgroup C_5 of $\text{Sym}(5)$.

$$C_5 = \{\text{id}, (1\,4\,2\,5\,3), (1\,2\,3\,4\,5), (1\,5\,4\,3\,2), (1\,3\,5\,2\,4)\}.$$

•

Definition 3.2.5. Let G be a group, $H \leq G$ and $\pi \in G$. A *right coset* $H\pi$ of H in G is the set of elements $H\pi = \{h\pi : h \in H\}$. Similarly $\pi H = \{\pi h : h \in H\}$ is a *left coset* of H in G. ◇

Theorem 3.2.6. *Let G be a group, $H \leq G$. The right (left) cosets of H in G form a partition of G. All cosets of H in G have the same size.*

Let G be a group, $H \leq G$. The *index* of H in G is the number of disjoint right (left) cosets of H in G. The number of left cosets is equal to the number of right cosets of H in G. We shall denote the index of H in G by $[G : H]$

Example 3.5. Consider $C_3 = \{\text{id}, (1\,3\,2), (1\,2\,3)\} < \text{Sym}(3)$. The right cosets of C_3 in $\text{Sym}(3)$ are given by $C_3\,\text{id} = C_3 = \{\text{id}, (1\,3\,2), (1\,2\,3)\}$ and $C_3\,(2\,3) = \{(1\,2), (2\,3), (1\,3)\}$. •

Theorem 3.2.7 (Lagrange)**.** *Let G be a finite group and $H \leq G$. Then $|H|$ divides $|G|$ and $[G : H] = |G| \,/\, |H|$.*

Definition 3.2.8. Let G be a group, $H \leq G$. A subset $T \subseteq G$ is a *right (left) transversal* of H in G if T contains exactly one element from each right (left) coset of H in G. $\diamond$

Elements of T are sometimes called *coset representatives*. We have that G is equal to the disjoint union of all right cosets $H\,t$ with coset representatives $t \in T$. If π is an element of G then there is a unique coset representative $t \in T$ and a unique element h of H such that $\pi = h\,t$.

Example 3.6. Consider $C_4 = \{id, (1\,2\,3\,4), (1\,3)\,(2\,4), (1\,4\,3\,2)\} < \mathrm{Sym}(4)$. The set

$$T = \{\mathrm{id}, (3\,4), (2\,4), (2\,3), (2\,3\,4), (2\,4\,3)\}$$

is a right transversal of C_4 in $\mathrm{Sym}(4)$. $\bullet$

Definition 3.2.9. Given a group $(G, *)$ and a non-empty set X. A binary operator $\cdot$ is called a *left action* of G on X if $\pi \cdot x \in X$, $\mathrm{id} \cdot x = x$ and $(\pi_1 * \pi_2) \cdot x = \pi_1 \cdot (\pi_2 \cdot x)$ for all π, π_1, $\pi_2 \in G$, and all $x \in X$ $\diamond$

Definition 3.2.10. Given a group $(G, *)$ and a non-empty set X. A binary operator $\cdot$ is called a *right action* of G on X if $x \cdot \pi \in X$, $x \cdot \mathrm{id} = x$ and $x \cdot (\pi_1 * \pi_2) = (x \cdot \pi_1) \cdot \pi_2$ for all π, π_1, $\pi_2 \in G$, and all $x \in X$. $\diamond$

We shall write $\pi\,x$ instead of $\pi \cdot x$ and use the exponential notation x^π instead of $x \cdot \pi$ (with $x \in X$, $\pi \in G$) whenever the operator $\cdot$ is clear from context. Throughout this text the following elementary group action will frequently occur.

Definition 3.2.11. The natural (right) action of a permutation group $G \leq \mathrm{Sym}(X)$ on X is defined by $x \cdot \pi = x^\pi$ for every $x \in X$ and $\pi \in G$. $\diamond$

Definition 3.2.12. Consider the right (or left) action of a group G onto a set X. For each $x \in X$, the set $x^G = \{x^\pi : \pi \in G\}$ (or $G\,x = \{\pi\,x : \pi \in G\}$) is called the *orbit* of x under G. $\diamond$

Example 3.7. Consider the natural right action of $G = \langle(1\,4\,3\,2)\,, (2\,4)\rangle$ on the set $X = \{1\ \ldots, 4\}$. Then $2^G = \{1, 2, 3, 4\}$. $\bullet$

The definition of a group guarantees that the set of orbits of X under the right (left) action of G form a partition of X. The associated equivalence relation $\sim$ is defined by saying $x \sim y$ with $x, y \in X$ if and only if there exists a $\pi \in G$ with $x^\pi = y$ (or $\pi\,x = y$). The orbits are then the equivalence classes under $\sim$.

We write $G \setminus\!\!\setminus X$ for the set of all orbits of X under the right (or left) action of a group G.

Example 3.8. Let $g_1 = (1\,4\,2\,3)\,(5\,11\,10\,6)$, $g_2 = (1\,4\,5\,11\,6\,10\,3\,2)\,(8\,9)$ and $g_3 = (1\,4\,5\,11\,6\,10\,3\,2)\,(7\,8)$. Consider the action of $G = \langle g_1, g_2, g_3\rangle$ on the set $X = \{1\,\ldots, 11\}$. Then $G \setminus\!\!\setminus X = \{\{1, 2, 3, 4, 5, 6, 10, 11\}, \{7, 8, 9\}\}$. •

Definition 3.2.13. Consider the right (or left) action of a group G on a set X. For each $x \in X$, the set $G_x = \{\pi \in G : x^\pi = x\}$ (or $G_x = \{\pi \in G : \pi\, x = x\}$) is called the *stabilizer* of x in G. ◇

It is easily seen that G_x is a subgroup of G.

Example 3.9. Consider the action of $G = \langle (1\,4\,3\,2)\,, (2\,4)\rangle$ on the set $X = \{1\,\ldots, 4\}$. Then $G_2 = \{\mathrm{id},\, (1\,3)\}$. •

For each $x \in X$, the orbit x^G (or $G\,x$) and the corresponding stabilizer subgroup G_x are connected by the following result:

Theorem 3.2.14. *Consider the right (or left) action of a group G on a non-empty set X and let T be a right (or left) transversal of the stabilizer G_x in G with $x \in X$. Then the mapping $t \mapsto x^t$ (or $t \mapsto t\,x$) for all $t \in T$ is a bijection of the transversal T onto the orbit x^G (or $G\,x$).*

A bijection tells us that two sets have the same size, hence we find immediately the following consequence:

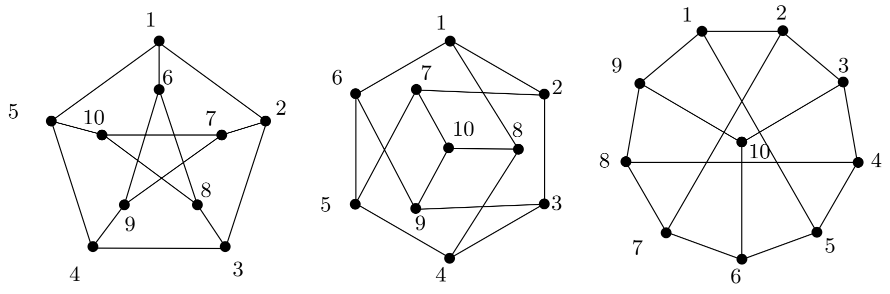
Theorem 3.2.15 (Orbit-Stabilizer)**.** *Let the group G act right (or left) on the set X and let $x \in X$, then $|x^G| = [G : G_x]$ (or $|G\,x| = [G : G_x]$).*

Example 3.10. Consider the action of $G = \langle (1\,4\,3\,2)\,, (2\,4)\rangle$ on $X = \{1\,\ldots, 4\}$. Consider the stabilizer $G_2 = \{\mathrm{id},\, (1\,3)\}$. Then $2^G = \{1, 2, 3, 4\}$ and a right transversal of G_2 in G is $T = \{\mathrm{id}, (1\,2\,3\,4),\, (1\,2)\,(3\,4),\, (2\,4)\}$. •

3.2.2 Isomorphism in terms of induced group actions

The isomorphism relation associated with virtually all combinatorial objects — among which the combinatorial objects covered in this text — can be naturally characterized in terms of group actions. Recall the definition of graph isomorphism given by Definition 2.1.3. The following example illustrates how we can capture this particular isomorphism relation in terms of induced group actions.

Example 3.11. Consider the three isomorphic graphs I, J and K, respectively shown from left to right in the figure below.



Since the isomorphic graphs I, J and K have the same vertex set, each isomorphism of one of these graphs onto another must correspond to a permutation of that vertex set. E.g., $\pi = (3\,7)\,(4\,5\,6\,8\,9) \in \mathrm{Sym}(10)$ is an isomorphism of I onto J, while $\sigma = (6\,9\,8\,10) \in \mathrm{Sym}(10)$ is an isomorphism of J onto K. We leave it for the reader to verify that $\pi \cdot I = J$, $\sigma \cdot J = K$ and $\pi\,\sigma \cdot I = K$. •

Consider $\mathrm{Sym}(n)$, then its *induced action* on the finite set $\mathcal{G}_n$ of graphs of order n can be defined by $g \cdot X$ such that $V(g \cdot X) = V(X)$ and

$$E(g \cdot X) = \{\{x, y\} \,:\, \{x^g, y^g\} \in E(X)\}$$

for every $X \in \mathcal{G}_n$ and every $g \in \mathrm{Sym}(n)$. In other words, x and y are joined in $g \cdot X$ if and only if x^g and y^g are joined in X. Clearly $(g\,h) \cdot X = g \cdot (h \cdot X)$ for every $X \in \mathcal{G}_n$ and every $g, h \in \mathrm{Sym}(n)$. After all, we have $V(g \cdot (h \cdot X)) = V(g\,h \cdot X) = V(X)$ and

$$\begin{aligned} E(g \cdot (h \cdot X)) &= \{\{x, y\} \,:\, \{x^g, y^g\} \in E(h \cdot X)\} \\ &= \{\{x, y\} \,:\, \{(x^g)^h, (y^g)^h\} \in E(X)\} \\ &= E(g\,h \cdot X)) \end{aligned}$$

In terms of this induced group action, two graph graphs $X, Y \in \mathcal{G}_n$ are isomorphic if there exists a $g \in \mathrm{Sym}(n)$ such that $g \cdot X = Y$. The associated permutation g is called an isomorphism of X onto Y.

In a more general context, we are equipped with a finite set of combinatorial objects Δ and a group G whose induced actions on Δ correspond to all the isomorphisms between the objects of Δ. More precisely, for every $\pi \in G$ and every $X \in \Delta$ there corresponds an object $\pi \cdot X$, obtained by the induced action of π on X. Two objects $X, Y \in \Delta$ are isomorphic, and we write $X \cong_G Y$, if there exists a $\pi \in G$ such that $\pi \cdot X = Y$. If the group G acting on Δ is clear from context, we shall write $X \cong Y$ instead. If there exists a $\sigma \in G$ such that $\sigma \cdot X = X$, then σ is called an *automorphism* of X. The set of all automorphisms of X forms a group, called the *automorphism group* of X. The automorphism group of X shall be denoted by $\operatorname{Aut} X$. In terms of a finite set of combinatorial objects Δ, the orbit of $X \in \Delta$ under G translates to

$$G\,X = \{\pi \cdot X : \pi \in G\} = \{Y \in \Delta : X \cong_G Y\}$$

that is, the set of combinatorial objects of Δ which are isomorphic to X. Moreover, the stabilizer of X in G translates to

$$G_X = \{\pi \in G : \pi \cdot X = X\} = \operatorname{Aut} X$$

that is, the automorphism group of X.

Example 3.12. Consider the graph I as defined in Example 3.11, then $\operatorname{Aut} I = \langle (1\,2\,3\,4\,5)\,(6\,7\,8\,9\,10), (1\,2\,3\,4\,9\,6)\,(5\,7\,8) \rangle$ and $|\operatorname{Aut} I| = 120$. •

Explicitly checking whether two objects $X, Y \in \Delta$ are isomorphic, requires finding an isomorphism of X onto Y, or vice versa, an isomorphism of Y onto X. However, in practice it is customary to check whether $X \cong_G Y$ by comparing the *canonical representatives* of X and Y with respect to some *canonical representative map*:

Definition 3.2.16. A *canonical representative map* for the induced action of G on Δ is a function $\varphi : \Delta \to \Delta$ such that $\varphi(X) \cong_G X$ for every $X \in \Delta$ and $X \cong_G Y$ implies $\varphi(X) = \varphi(Y)$ for every $X, Y \in \Delta$. ◇

Given such a canonical representative map φ, an object $X \in \Delta$ is said to be in *canonical form* if $\varphi(X) = X$. Moreover, $\varphi(X)$ is said to be the *canonical representative* or *canonical labeled object* of $G\,X$, the orbit of X under G. Since $X \cong_G Y$ if and only if $\varphi(X) = \varphi(Y)$, checking whether X and Y are isomorphic essentially comes down to testing whether their canonical representatives are identical.

Definition 3.2.17. A *canonical labeling map* for the induced action of G on Δ is a function $\vartheta : \Delta \to G$ such that

$$\vartheta(\pi X)\,\pi X = \vartheta(X)\,X$$

for every $\pi \in G$ and every $X \in \Delta$. ◇

The isomorphism $\vartheta(X)$ of X onto $\vartheta(X)\,X$ is said to be a *canonical labeling*. A canonical representative map and a canonical labeling map are closely related. On the one hand, given a canonical labeling map ϑ for the induced action of G on Δ , then the map $\varphi : \Delta \to \Delta$ such that

$$\varphi(X) = \vartheta(X)\,X$$

for every $X \in \Delta$ is the canonical representative map. On the other hand, given a canonical representative map φ for the induced action of G on Δ, then any map $\vartheta : \Delta \to G$ such that

$$\vartheta(X) \in \{\pi \in G : \pi X = \varphi(X)\}$$

is a canonical labeling map.

Definition 3.2.18. An *invariant* for the induced action of G on Δ is a function ξ with domain Δ such that $\xi(X) = \xi(Y)$ if $X \cong_G Y$ for every $X, Y \in \Delta$. ◇

Definition 3.2.19. A *certificate* is an invariant ξ for the induced action of G on Δ such that $\xi(X) = \xi(Y)$ if and only if $X \cong_G Y$ for every $X, Y \in \Delta$. ◇

An *invariant* is an attribute of the objects of Δ which remains constant under the induced action of G on Δ. Since $\xi(X) \neq \xi(Y)$ implies $X \ncong_G Y$, we may apply invariants to distinguish between non-isomorphic objects of Δ. However, it is not necessarily so that $X \ncong_G Y$ implies $\xi(X) \neq \xi(Y)$. This is true if and only if ξ is a certificate.

Example 3.13. The clique number of graph G of order n is a *graph invariant* under the induced action of $\mathrm{Sym}(n)$ on $\mathcal{G}_n$, but not a certificate. •

3.3 Isomorphism rejection

A central theme in virtually all exhaustive generation algorithms is the detection and elimination of isomorphic combinatorial objects. In general, let Δ denote the set of combinatorial objects on which a finite group G induces actions, then the aim of isomorphism rejection essentially is to generate exactly one representative from each orbit of $G \bbslash \Delta$. Moreover, failure to remove isomorphic copies results in the traversal of superfluous regions of the associated search tree, proportional with the number of isomorphic combinatorial objects. Even when considering relatively small combinatorial objects, this would render the exhaustive generation process inefficient or make it even completely infeasible. Furthermore, the application of isomorphism rejection techniques avoids traversing regions of the search tree which are identical in terms of the exhaustive generation process but which do not contain leaf nodes corresponding to combinatorial objects of the set Δ.

McKay [70] divides advanced algorithms for isomorph-free exhaustive generation into three categories. A slightly modified classification appears in [8, 65]. McKay distinguishes between the following approaches:

Orderly generation In this approach there is exactly one canonical labeled object in each orbit and that is the orbit representative being generated. It is customary to choose a canonical form such that specific subobjects of canonical labeled objects are also in canonical form. The term orderly stems from the fact that objects are generated subject to some total order on them. Orderly algorithms were independently introduced by Read [87] and Faradžev [41]. Applications of this orderly approach can be found for the classification of many combinatorial objects including designs and their resolutions [33, 34, 35, 82, 83], cubic graphs [9], regular graphs [79, 80], one factorizations [37] and fullerenes [10].

Canonical augmentation This approach was introduced by McKay in [70]. A slightly simplified version, called *weak canonical augmentation*, appears in [65]. In this approach, combinatorial objects are constructed in a canonical way, instead of the combinatorial objects being canonical themselves. Essentially an axiomatic model is imposed on the search tree such that two isomorphic nodes are constructed by a sequence of isomorphic subobjects. Isomorph-free generation is obtained by systematically rejecting isomorphic siblings during traversal. Applications of this (weak) canonical augmentation approach can be found for the classification of many combinatorial objects including Steiner triple systems [63], posets [11], Latin squares, quasigroups and loops [75].

Homomorphism principle This approach was introduced by Grüner, Laue and

Meringer in [51]. In this approach combinatorial objects are constructedalong a path of combinatorially determined subobjects, using elementary group computations which describe the relationship between the automorphism structure of consecutive subobjects [70]. Applications of this homomorphism principle approach can be found for the exhaustive generation of many combinatorial objects including t-designs [89], polyhex hydrocarbons [18] and molecular graphs [50].

In studying tailored isomorph-free exhaustive generation algorithms for association schemes and in particular strongly regular and distance regular graphs, we shall concentrate [76] on orderly generation algorithms. For a detailled introduction on canonical augmentation and the homomorphism principle we refer the reader to [8, 65, 70] and [8, 51, 65], respectively.

In a more general setting, orderly generation algorithms can be described in detail in terms of constraint networks and search trees. Consider a constraint network $\mathcal{R} = (X, D, C)$ with n variables, where the finite set of leaf nodes at depth n in the associated search tree corresponds to a finite set Δ of combinatorial objects on which a finite group G induces actions. So as to describe orderly generation, we need to extend the induced action of G to the finite set $N_{\mathcal{R}}$, consisting of all nodes – partial and complete – of the search tree.

Consider a canonical representative map φ for the induced action of G on the set $N_{\mathcal{R}}$. In an orderly approach, it is customary to prune all nodes in the search tree which are not in canonical form with respect to φ. However, in order to generate exactly one representative from each orbit of $G \backslash\backslash \Delta$, the canonical representative map φ must satisfy $\varphi(X) \in N_{\mathcal{R}}$ for every $X \in N_{\mathcal{R}}$ and if $\varphi(Y) = Y$ then also $\varphi(Z) = Z$ for every $Y, Z \in N_{\mathcal{R}}$ with Z the parent of Y.

As a result of these requirements, only one representative of each orbit of $G \backslash\backslash \Delta$ is generated. Indeed, it is straightforward to see that at most one representative is generated for each orbit, since only canonical representatives are considered as solutions of the constraint network $\mathcal{R} = (X, D, C)$. Moreover, for every $X \in \Delta$ we have that all nodes on the path from the root of the search tree to the leaf node $\varphi(X)$ are in canonical form. Hence at least one representative of each orbit is generated.

A pseudo code description of a general orderly algorithm is given in Algorithm 3.2. The same control strategy as in Algorithm 3.1 is applied, that is, at each step in the recursion the uninstantiated variable with the smallest index is selected.

Algorithm 3.2 General orderly algorithm for a constraint network $\mathcal{R} = (X, D, C)$ with n variables subject to a canonical representative map φ for the induced action of G on $N_{\mathcal{R}}$.

```
procedure search( )
1: orderly(( ),0)

procedure orderly(($d_1, \ldots, d_k$): instantiation, $k$: int))
1: if (($d_1, \ldots, d_k$) is in canonical form then
2:    if $k = n$ then
3:       report ($d_1, \ldots, d_n$) as a solution
4:    else
5:       for all $d_{k+1} \in D_{k+1}$ do
6:          if ($d_1, \ldots, d_{k+1}$) is feasible then
7:             orderly(($d_1, \ldots, d_{k+1}$),k+1)
```

Below we give an example to illustrate the preceding general discussion of the orderly approach in terms of constraint networks and search trees.

Example 3.14. Recall the constraint network $\mathcal{R} = (X, D, C)$ which describes the corresponding exhaustive generation of the set Δ of all regular graphs of order 4 and degree 2 as outlined in Example 3.1–3.2. The group Sym(4) induces an action on Δ, and at the same time on the set $N_{\mathcal{R}}$ of 4×4 matrices of the corresponding recursion tree. More precisely, the induced action of Sym(4) on $N_{\mathcal{R}}$ can be defined for every $M \in N_{\mathcal{R}}$, $\pi \in \mathrm{Sym}(4)$ by

$$(\pi \cdot M)_{x,y} = (M)_{x^\pi, y^\pi},$$

for all x, $y \in \{1, \ldots, 4\}$. Hence, two matrices of $N_{\mathcal{R}}$ are isomorphic if and only if the one can be obtained from the other by permuting its rows and columns at the same time.

Define the ordering $? < 0 < 1$, and associate with each $M \in N_{\mathcal{R}}$ a 6-tuple over $\{?, 0, 1\}$, which is formed by concatenating the upperdiagonal columns of M. The control strategy applied in Example 3.2 is defined in such a way that the 6-tuples associated with each node in the search tree occur in lexicographic order. A matrix $M \in N_{\mathcal{R}}$ is in canonical form, when the 6-tuple associated with each matrix in the orbit of M under $\mathrm{Sym}(4)$ is lexicographically smaller or equal than the 6-tuple associated with M.

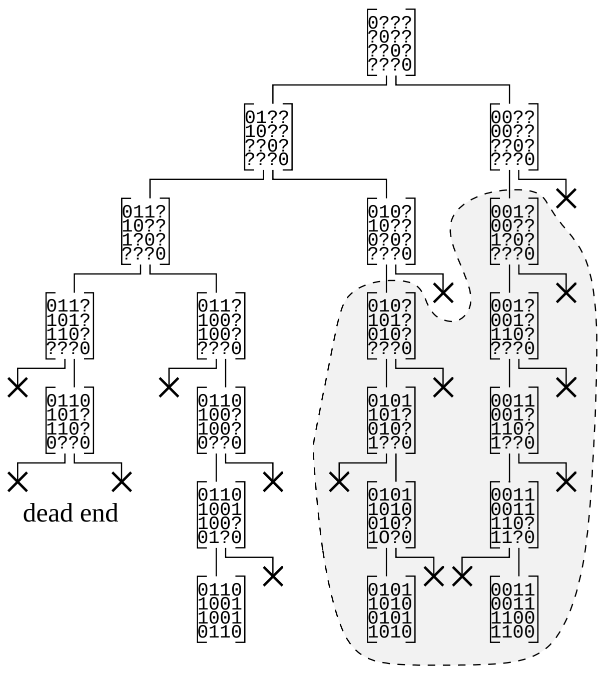
The canonical representative of every $M \in N_{\mathcal{R}}$ occurs as a node in the search tree. Moreover, for every nonroot node which is in canonical form, it holds that its parent is also in canonical form. Hence under the assumption of the above canonical form, Figure 3.2 corresponds to the search tree traversed by Algorithm 3.2. The subtrees shown in the light gray areas are discarded by the algorithm. The leftmost subtree can be pruned since

$$\pi \cdot \begin{pmatrix} 0 & 1 & 0 & ? \\ 1 & 0 & 1 & ? \\ 0 & 1 & 0 & ? \\ ? & ? & ? & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & ? \\ 1 & 0 & 0 & ? \\ 1 & 0 & 0 & ? \\ ? & ? & ? & 0 \end{pmatrix}$$

with $\pi = (1\,2\,3) \in \mathrm{Sym}(4)$, while the rightmost subtree can be pruned since

$$\sigma \cdot \begin{pmatrix} 0 & 0 & 1 & ? \\ 0 & 0 & ? & ? \\ 1 & ? & 0 & ? \\ ? & ? & ? & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & ? \\ 1 & 0 & ? & ? \\ 0 & ? & 0 & ? \\ ? & ? & ? & 0 \end{pmatrix}$$

with $\sigma = (2\,3) \in \mathrm{Sym}(4)$. Indeed, the matrices associated with the root of these subtrees are not in canonical form. •



Figuur 3.2: Search tree corresponding to the isomorph-free exhaustive generation of all regular graphs of order 4 and degree 2.

4 Isomorph-free generation of association schemes

”Real stupidity beats artificial intelligence every time.”[T. PRATCHETT, HOGFATHER]

In this chapter we concentrate on tailored, isomorph-free exhaustive generation algorithms for the classification of d-class association schemes, and strongly regular and distance regular graphs in particular. Our focus is on developing and improving orderly generation algorithms which provide a complete classification for a given parameter set and which ultimately turn out to be fast enough in practice to yield new classification results of combinatorial interest.

4.1 Exhaustive generation

In the most general context of association schemes, we are interested in the classification, up to isomorphism, of the set of all d-class association schemes on $V = \{1, \ldots, n\}$, given a set of intersection matrices[1] $\{L_0, \ldots, L_d\}$. Recall that the set of all such d-class association schemes shall be denoted by $\mathcal{X}_{L_0 \ldots L_d}$. This set $\mathcal{X}_{L_0 \ldots L_d}$ can be expressed as the set of solutions of a constraint network $\mathcal{R} = (X, D, C)$ consisting of a finite set of $(n^2 - n)/2$ variables

$$X = \{x_{ij} : 1 \leq i < j \leq n\} \tag{4.1}$$

together with a corresponding set of domains D

$$D = \{D_{ij} : 1 \leq i < j \leq n\}. \tag{4.2}$$

For now, all domains D_{ij} associated with the variables x_{ij} are the same and equal to $\{1, \ldots, d\}$.

When translating this classification problem into the constraint network $\mathcal{R}$, we face the challenge to identify mathematical properties of the d-class association schemes in $\mathcal{X}_{L_0 \ldots L_d}$, which may serve as a basis for the set of constraints C of the constraint network. These constraints should guarantee that for each solution of $\mathcal{R}$, the resulting matrix M as defined in (4.3) — with each variable of X instantiated — corresponds to the relation matrix M_Ω of a unique d-class association scheme $\Omega \in \mathcal{X}_{L_0 \ldots L_d}$.

$$M = \begin{pmatrix} 0 & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ x_{1,2} & 0 & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \cdots & \cdots \\ x_{1,n-1} & x_{2,n-1} & \vdots & 0 & x_{n-1,n} \\ x_{1,n} & x_{2,n} & \vdots & x_{n-1,n} & 0 \end{pmatrix} \tag{4.3}$$

The exhaustive generation algorithm, which we shall develop throughout this chapter, initially starts with a matrix M where all non diagonal matrix entries are still uninstantiated. (Recall that we write $M_{ij} =?$ when the variable at the corresponding position is uninstantiated.) Each upperdiagonal matrix entry M_{xy} — and at the same time its symmetric counterpart M_{yx} — is systematically recursively instantiated with each value from the associated domain D_{xy}. Nodes at depth k in the corresponding recursion tree represent partially instantiated

[1]It is customary to express the intersection matrices of strongly regular and distance regular in terms of (v, k, λ, μ) parameter sets and intersection arrays, respectively.

matrices M where exactly k upperdiagonal entries of M are already filled in. During this recursive traversal, we may prune nodes of the recursion tree which correspond to partially instantiated matrices M which are not feasible, that is, which violate at least one of the constraints listed in C. The constraint set C consists of constraints which are defined based on both *combinatorial* as well as *algebraic* properties of the d-class association schemes in $\mathcal{X}_{L_0 \ldots L_d}$.

4.1.1 Combinatorial constraints

Several combinatorial properties of the d-class association schemes in $\mathcal{X}_{L_0 \ldots L_d}$ may serve as a basis for some of the constraints in C. The following constraints, derived from the defining axioms (2.4) of a d-class association scheme, are used as a pruning technique in our exhaustive generation algorithms. Recall that $N_{\mathcal{R}}$ denotes the set of all nodes, and thus equivalently, all partially and completely instantiated matrices M, which are considered during exhaustive generation.

Constraint 4.1.1 (Regularity constraint). *Let $M \in N_{\mathcal{R}}$, let $x \in V$ and $1 \leq i, j \leq d$. Define*

$$Z_i^x = \{z \in V : M_{xz} = i\} \quad \text{and} \quad k_i^x = |Z_i^x|. \tag{4.4}$$

Then $k_i^x \leq k_i = p_{ii}^0$.

The value k_i^x is called a *tentative valency*.

Constraint 4.1.2 (Intersection constraint). *Let $M \in N_{\mathcal{R}}$, let $x, y \in V$ and $1 \leq i, j, l \leq d$. Define*

$$Z_{ij}^{xy} = \{z \in V : M_{xz} = i \text{ and } M_{zy} = j\} \quad \text{and} \quad p_{ij}^{xy} = |Z_{ij}^{xy}|. \tag{4.5}$$

Then $p_{ij}^{xy} \leq p_{ij}^l$ if $M_{xy} = l$.

The value p_{ij}^{xy} is called a *tentative intersection number*.

Constraint 4.1.3 (Tentative intersection constraint). *Let $M \in N_{\mathcal{R}}$, let $x, y \in V$ and $1 \leq i, j \leq d$, such that $M_{xy} =?$. Then $p_{ij}^{xy} \leq \max_{l=1}^{d} p_{ij}^l$.*

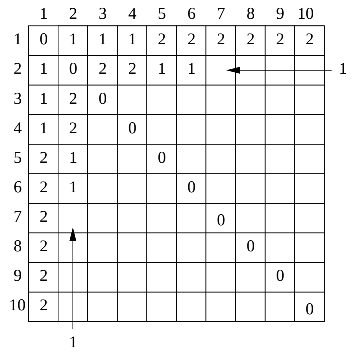
These combinatorial constraints can be used to prune the recursion tree as follows. Let $i \in \{1, \ldots, d\}$. If during the traversal of the recursion tree we encounter a partially instantiated matrix $M \in N_{\mathcal{R}}$, we may reject M and prune the corresponding branch of the recursion tree,

- if at least one of its rows or columns contains more than $k_i = p^0_{ii}$ entries with value i;
- if the number of elements r for which the matrix entry $M_{pr} = i$ and the matrix entry $M_{rq} = j$ is more than p^k_{ij}, where the matrix entry $M_{pq} = k$;
- if the matrix entry M_{pq} is uninstantiated and the number of elements r for which the matrix entry $M_{pr} = i$ and the matrix entry $M_{rq} = j$ is more than $\max^d_{k=1} p^k_{ij}$.

Indeed, in each case M cannot possibly be extended to a completely instantiated matrix which corresponds to the relation matrix M_Ω of a d-class association scheme $\Omega \in \mathcal{X}_{L_0 \ldots L_d}$.

Note that in order to check the regularity constraints in practice, not every possible element $x \in V$ needs to be considered, but only those elements for which x is a row or column number of the matrix entry of M that was instantiated in the last step of the recursion. Similarly, in order to check the (tentative) intersection constraints, not every possible pair $x, y \in V$ needs to be considered, but only those pairs for which x or y is a row or column number of the matrix entry of M that was instantiated in the last step of the recursion.

Example 4.1. Consider a two-class association scheme with $p^1_{2,1} = 2$, $k_1 = p^0_{1,1} = 3$ and a partially instantiated matrix M as depicted below.



	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	2	2	2	2	2	2
2	1	0	2	2	1	1				
3	1	2	0							
4	1	2		0						
5	2	1			0					
6	2	1				0				
7	2						0			
8	2							0		
9	2								0	
10	2									0

The extension of this matrix M by instantiating the matrix entry $M_{2,7}$ with value 1 (and the same value at the symmetric position) can be rejected because then on the one hand $k_1^2 = 4 > k_1 = 3$ and on the other hand $p_{2,1}^{1,2} = 3 > p_{2,1}^1 = 2$ with matrix entry $M_{1,2} = 1$, which violates Constraint 4.1.1 and 4.1.2, respectively. ●

4.1.2 Combinatorial look-ahead strategies

Basic backtracking has the problem that some of the domain values are *inconsistent* with the matrix M as it is instantiated thus far: adding them to the matrix at the given position would violate some of the given combinatorial constraints. This can be avoided by applying the following look-ahead strategy. For each matrix entry $M_{x,y}$ we keep track of the domain of values which can safely be instantiated at that position. After every recursion step we examine the domain for each entry which at that point has not yet been instantiated. Any value for that entry which is found to violate at least one combinatorial constraint is then (temporarily) removed from the domain of that entry. Note that the application of these domain reductions induces an additional constraint: whenever a domain for an uninstantiated entry becomes empty, then the matrix M cannot be further extended.

In practice we implement this concept in a somewhat different manner: instead of examining every uninstantiated entry at each step, we only select entries for domain reduction according to the following rules :

Look-ahead 4.1.1. *Let $M \in N_{\mathcal{R}}$. Let $x \in V$, $1 \leq i, j \leq d$. If $k_i^x = k_i$, then $w \in V \setminus Z_i^x$ cannot satisfy $M_{wx} = i$.*

As before, not every possible element $x \in V$ needs to be considered in practice, but only those elements for which x is a row or column number of the matrix entry of M that was instantiated in the last step. Moreover, only the domains of still uninstantiated matrix entries should be reduced.

Look-ahead 4.1.2. *Let $M \in N_{\mathcal{R}}$. Let $x, y \in V$, $1 \leq i, j \leq d$. Then the following rules must hold:*

1. *If $M_{xy} = k$ and $p_{ij}^{xy} = p_{ij}^k$, then $w \in V \setminus Z_{ij}^{xy}$ such that $M_{wx} = i$ cannot*

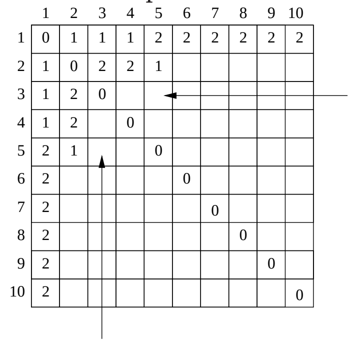
satisfy $M_{wy} = j$ at the same time. Similarly, any $w \in V \setminus Z_{ij}^{xy}$ for which $M_{wy} = j$ cannot also satisfy $M_{wx} = i$.

2. *If $M_{xy} =?$ and $p_{ij}^{xy} = \max_k p_{ij}^k$ then $w \in V \setminus Z_{ij}^{xy}$ such that $M_{wx} = i$ cannot satisfy $M_{wy} = j$ at the same time. Similarly, any $w \in V \setminus Z_{ij}^{xy}$ for which $M_{wy} = j$ cannot also satisfy $M_{wx} = i$.*

3. *If $p_{ij}^{xy} > p_{ij}^k$ for some $k, 1 \leq k \leq d$ then x, y cannot satisfy $M_{xy} = k$.*

Again, not every possible pair $x, y \in V$ needs to be considered, but only those pairs for which x or y is a row or column number of the matrix entry of M that was instantiated in the last step. As before, only the domains of still uninstantiated matrix entries should be reduced.

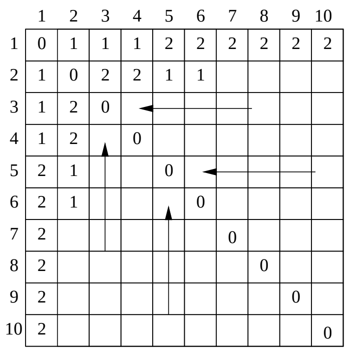
Example 4.2. Consider a two-class association scheme with $p_{11}^2 = 1$ and a partially instantiated matrix M as depicted below.



	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	2	2	2	2	2	2
2	1	0	2	2	1					
3	1	2	0							
4	1	2		0						
5	2	1			0					
6	2					0				
7	2						0			
8	2							0		
9	2								0	
10	2									0

Clearly $M_{2,3} = 2$ and $p_{11}^{2,3} = p_{11}^2$. Hence applying the first domain reduction rule of Look-ahead 4.1.2, reduces the domain of the matrix entry $M_{3,5}$ to $\{2\}$. ●

Example 4.3. Consider a two-class association scheme with $p_{11}^1 = 0$, $p_{11}^2 = 1$ and a partially instantiated matrix M as depicted below.



	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	2	2	2	2	2	2
2	1	0	2	2	1	1				
3	1	2	0							
4	1	2		0						
5	2	1			0					
6	2	1				0				
7	2						0			
8	2							0		
9	2								0	
10	2									0

Clearly $M_{5,6} =?$ and $p_{11}^{5,6} = 1 > p_{11}^{1}$. Similarly, $M_{3,4} =?$ and $p_{11}^{3,4} = 1 > p_{11}^{1}$. Hence applying the third domain reduction rule of Look-ahead 4.1.2, reduces the domains of the matrix entries $M_{5,6}$ and $M_{3,4}$ to $\{2\}$. •

4.1.3 Dynamic variable ordering

Experiments show that the order in which matrix entries of M are chosen for instantiation substantially influences the running time of the backtrack algorithm. We have tried several heuristics for selecting an optimal ordering. In some cases an ordering was determined in advance and remained fixed during the algorithm, in other cases the next entry to be considered was chosen dynamically at each step. We obtained the best results by extending each node in the recursion tree with the matrix entry whose remaining domain had the smallest cardinality at that time.

However, the orderly isomorphism reduction techniques which we shall introduce in Section 4.2 require a fixed order in which matrix entries are chosen for instantiation. Nevertheless, this domain information can still be used to deviate from this fixed order whenever a domain of an uninstantiated matrix entry has size 1. In other words, when a domain value is forced for a certain uninstantiated matrix entry, we instantiated that matrix entry first. Note that a *dynamic variable ordering* strategy like this implies that the order of instantiation does not need to be the same in different branches of the recursion tree.

4.1.4 Algebraic constraints

Also several *algebraic properties* related to the Bose-Mesner algebra of the d-class association schemes in $\mathcal{X}_{L_0 \dots L_d}$ may serve as a basis for some of the constraints in C. First, recall that a *principal submatrix* of a square symmetric $n \times n$ matrix is any $m \times m$ submatrix obtained by deleting $n - m$ rows and corresponding columns, while a *leading principal submatrix* is any $m \times m$ submatrix obtained by deleting the last $n - m$ rows and columns.

In Chapter 2 we have shown that the eigenmatrix P and the dual eigenmatrix Q of an association schemes can be computed from its intersection matrices $L_0, \dots, L_d$. Hence P and Q are the same for all elements of $\mathcal{X}_{L_0 \dots L_d}$. The constraints which we shall introduce below, are based on the properties of the minimal idempotents E_i of the schemes. In particular

- E_i is positive semidefinite (Lemma 2.4.3) and hence each principal submatrix of E_i is positive semidefinite.
- E_i has rank Q_{0i}, and hence each principal submatrix of E_i has rank at most Q_{0i}.

Also recall that the entry of E_i at position x, y is equal to $\frac{Q_{ki}}{n}$, when x and y are k-th associates, by (2.29).

The following constraints, derived from these algebraic properties of the minimal idempotents, are used as a pruning technique in our exhaustive generation algorithms. For each partially instantiated matrix $M \in N_{\mathcal{R}}$ and each $i \in \{0, \dots, d\}$, these constraints are expressed in terms of a closely related $n \times n$ matrix M_{E_i} with entries

$$(M_{E_i})_{xy} = \begin{cases} \frac{1}{n} Q_{ki} & \text{if } M_{xy} = k \\ ? & \text{if } M_{xy} = ? \end{cases} \tag{4.6}$$

for every $x, y \in V$ with $k \in \{0, \dots, d\}$. Note that when M is completely instantiated and corresponds to a relation matrix M_Ω of a d-class association scheme $\Omega \in \mathcal{X}_{L_0 \dots L_d}$, then the related matrix M_{E_i} is identical to the minimal idempotent E_i of Ω.

Constraint 4.1.4 (Positive semidefinite constraint). *Let* $0 \leq i \leq d$ *and let* $M \in$

$N_{\mathcal{R}}$ with corresponding matrix M_{E_i}. Then any completely instantiated principal submatrix of M_{E_i} must be positive semidefinite.

Constraint 4.1.5 (Rank constraint). *Let $0 \leq i \leq d$ and let $M \in N_{\mathcal{R}}$ with corresponding matrix M_{E_i}. Then any completely instantiated principal submatrix of M_{E_i} must have rank at most Q_{0i}.*

In other words, let $1 \leq i \leq d$. If during the traversal of the recursion tree we encounter a partially instantiated matrix $M \in N_{\mathcal{R}}$, we may reject M and prune the corresponding branch of the recursion tree, if any principal submatrix of the related matrix M_{E_i} — with all its matrix entries filled in — is not positive semidefinite or has a rank more than Q_{0i}. Indeed, M cannot possibly extended to a completely instantiated relation matrix M' since the corresponding matrix M'_{E_i} cannot be equal to the minimal idempotent E_i of a d-class association scheme in $\mathcal{X}_{L_0 \ldots L_d}$. Checking both algebraic constraints during traversal is conceptually straightforward, however incorporating both algebraic constraints in an explicit generation algorithm turns out to be complicated in practice. A more detailed discussion on how both constraints are actually checked during traversal of the recursion tree is given in Section 4.3.

4.2 Isomorph-free generation

The set of solutions of the constraint network $\mathcal{R}$ outlined in the previous section, consists of all d-class association schemes of $\mathcal{X}_{L_0 \ldots L_d}$. However, as in Section 3.3, it is our objective to generate only one representative for each isomorphism class. In the particular case of d-class association schemes, the isomorphism relation can be expressed using the following action of $\mathrm{Sym}(n)$ on $\mathcal{X}_{L_0 \ldots L_d}$.

Definition 4.2.1. Let $i \in \{0, \ldots, d\}$. Let $\pi \in \mathrm{Sym}(n)$ and $\Omega \in \mathcal{X}_{L_0 \ldots L_d}$. Then $\pi \cdot \Omega$ is the association scheme of $\mathcal{X}_{L_0 \ldots L_d}$ such that (x^π, y^π) are i-th associates in Ω if and only if (x, y) are i-th associates in $\pi \cdot \Omega$ for every $x, y \in \{1, \ldots, n\}$. $\diamond$

Note that the relation matrix $M_{\pi \cdot \Omega}$ of $\pi \cdot \Omega$ satisfies

$$(M_{\pi \cdot \Omega})_{xy} = (M_\Omega)_{x^\pi, y^\pi}. \tag{4.7}$$

Note that this is indeed a left action, as

$$(M_{\pi\cdot(\sigma\cdot\Omega)})_{x,y} = (M_{\sigma\cdot\Omega})_{x^\pi,y^\pi} = (M_\Omega)_{x^{\pi\sigma},y^{\pi\sigma}}. \tag{4.8}$$

Two d-class association schemes Ω_1, $\Omega_2 \in \mathcal{X}_{L_0\ldots L_d}$ are called isomorphic if there exists a $\pi \in \mathrm{Sym}(n)$ such that $\pi \cdot \Omega_1 = \Omega_2$. In terms of this induced action, generating only one representative from each isomorphism class is equivalent with generating one representative from each orbit of $\mathrm{Sym}(n) \mathbin{\backslash\!\backslash} \mathcal{X}_{L_0\ldots L_d}$.

The isomorphism rejection techniques incorporated in our exhaustive generation algorithm are based on an orderly approach. Recall from Section 3.3 that in order to describe these techniques, we need to extend the action of $\mathrm{Sym}(n)$ to the finite set $N_\mathcal{R}$ consisting of all nodes of the search tree. These nodes correspond to partially and completely instantiated square symmetric $n \times n$ matrices M with zero-diagonal. Let $\pi \in \mathrm{Sym}(n)$, then $\pi \cdot M$ is a square symmetric $n \times n$ matrix with zero-diagonal such that

$$(\pi \cdot M)_{xy} = (M)_{x^\pi,y^\pi} \tag{4.9}$$

for every x, $y \in \{1,\ldots,n\}$. Hence two square symmetric $n \times n$ matrices with zero-diagonal — and thus similarly for two matrices of $N_\mathcal{R}$ — are called isomorphic if the one can be obtained from the other by permuting its rows and columns at the same time. Moreover, let $\Omega \in \mathcal{X}_{L_0\ldots L_d}$, then clearly

$$\pi \cdot M_\Omega = M_{\pi\cdot\Omega} \tag{4.10}$$

according to (4.7) and (4.9).

Integrating orderly isomorphism rejection techniques in an exhaustive generation algorithm requires selecting eligible canonical representatives in such a way that the applied control strategy, maximally constrains the partially instantiated matrices of $N_\mathcal{R}$. In this section we discuss two types of canonical representatives which we shall evaluate later in this chapter. Both types have in common that they are defined to be minimal in their orbits with respect to some lexicographic ordering. Subject to the type of canonical form at hand, the control strategy of the exhaustive generation algorithm shall be adapted in such a way that these extremal matrices of $N_\mathcal{R}$ turn up first during the traversal of the recursion tree. Furthermore both types have in common that their canonical representatives satisfy the same necessary condition — called lexical ordering — as outlined in [20, 40].

To introduce both types of canonical representatives we first need to define an ordering on tuples. Consider the k-tuples $a = (a_1, \ldots, a_k)$ and $b = (b_1, \ldots, b_k)$. Then we write

$$(a_1, \ldots, a_k) < (b_1, \ldots, b_k)$$

if and only if there exists an index $i \in \{0, \ldots, k-1\}$ such that

$$a_1 = b_1, \ldots, a_i = b_i \quad \text{and} \quad a_{i+1} < b_{i+1}.$$

The ordering defined in this way is called the *lexical ordering* on k-tuples. If a tuple a is defined over $\Sigma = \{0, 1, \ldots, d, ?\}$, then the undefined value ? is considered to be the maximal element of Σ. We shall denote the set of all square symmetric $n \times n$ matrices having an all zero diagonal and non-diagonal entries in Σ by $\mathcal{M}_{n,d}$.

4.2.1 Row order canonical form

Let $A \in \mathcal{M}_{n,d}$, then we define the *row order certificate* $\mathsf{cert_r}(A)$ of A to be the tuple over $\Sigma \setminus \{0\}$ of length $\frac{n\,(n-1)}{2}$ obtained by concatenating the upperdiagonal entries of A in a *row-by-row* order, that is,

$$\begin{aligned} \mathsf{cert_r}(A) \quad = \quad & (A_{1,2,\ldots,}A_{1,n}, A_{2,3,\ldots,}A_{2,n}, \ldots \\ & \qquad \ldots, A_{n-2,n-1}, A_{n-2,n}, A_{n-1,n}). \end{aligned} \tag{4.11}$$

Using the lexicographic ordering on tuples, we may define a total ordering on the set $\mathcal{M}_{n,d}$ as follows. Let $A, B \in \mathcal{M}_{n,d}$, then we define $A <_r B$ if and only if $\mathsf{cert_r}(A) < \mathsf{cert_r}(B)$. Consider the orbit $\mathrm{Sym}(n)\,A$ of $A \in \mathcal{M}_{n,d}$, then we define

$$\begin{aligned} \mathsf{canon_r}(A) &= \min_r\,(\mathrm{Sym}(n)\,A) \\ &= \min_r\,(\{\pi A \;:\; \pi \in \mathrm{Sym}(n)\}) \end{aligned} \tag{4.12}$$

The matrix $\mathsf{canon_r}(A)$ is *minimal* in its orbit and is called the *row order canonical representative* of A. If $A = \mathsf{canon_r}(A)$, we say that A is in *row order canonical form*[2].

[2]This standard type of canonical form was used e.g. in [9, 41, 79, 80, 87]

Example 4.4. Consider the matrices $A, B \in \mathcal{M}_{5,2}$ as shown below.

$$A = \begin{pmatrix} 0 & \mathbf{2} & \mathbf{1} & \mathbf{1} & \mathbf{2} \\ 2 & 0 & 2 & 1 & 1 \\ 1 & 2 & 0 & \mathbf{2} & \mathbf{1} \\ 1 & 1 & 2 & 0 & ? \\ 2 & 1 & 1 & ? & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & \mathbf{2} & \mathbf{1} & \mathbf{1} & \mathbf{2} \\ 2 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & \mathbf{2} & \mathbf{1} \\ 1 & 1 & 2 & 0 & ? \\ 2 & 2 & 1 & ? & 0 \end{pmatrix}$$

The corresponding row order certificates are

$$\begin{aligned} \mathsf{cert_r}(A) &= (\mathbf{2}, \mathbf{1}, \mathbf{1}, \mathbf{2}, 2, 1, 1, \mathbf{2}, \mathbf{1}, ?) \\ \mathsf{cert_r}(B) &= (\mathbf{2}, \mathbf{1}, \mathbf{1}, \mathbf{2}, 2, 1, 2, \mathbf{2}, \mathbf{1}, ?) \end{aligned}$$

and therefore $A <_r B$. •

Orderly generating one representative of each orbit of $\mathrm{Sym}(n) \mathbin{\backslash\!\backslash} \mathcal{X}_{L_0 \ldots L_d}$, subject to the canonical representative map $\mathsf{canon_r}$, is equivalent to generating the set of row order canonical representatives

$$\mathsf{rep_r}(\mathcal{X}_{L_0 \ldots L_d}) = \{\Omega \in \mathcal{X}_{L_0 \ldots L_d} \ : \ M_\Omega = \mathsf{canon_r}(M_\Omega)\} \qquad (4.13)$$

However, recall that in order to generate exactly one representative from each orbit of $\mathrm{Sym}(n) \mathbin{\backslash\!\backslash} \mathcal{X}_{L_0 \ldots L_d}$, the control strategy of the exhaustive generation should be defined in such a way that

$$\mathsf{canon_r}(A) \in N_{\mathcal{R}} \qquad (4.14)$$

for every $A \in N_{\mathcal{R}}$ and

$$\mathsf{canon_r}(C) = C \text{ if } \mathsf{canon_r}(B) = B \qquad (4.15)$$

for every B, $C \in N_{\mathcal{R}}$ with C the parent of the nonroot node B.

Definition 4.2.2. The node $B \in N_{\mathcal{R}}$ shall be called *row order compatible* if and only if $\mathsf{cert_r}(B)$ is of the form

$$\mathsf{cert_r}(B) = (a_1, \ldots, a_k, ?, \ldots, ?) \qquad (4.16)$$

for some $k \in \{1, \ldots, (n^2 - n)/2\}$ and $a_i \in \{1, \ldots, d\}$ for all $i \in \{1, \ldots, k\}$. B is called a *root node* if $k = 0$ in the above. If B is not a root node, then the unique matrix $C \in N_{\mathcal{R}}$ such that

$$\mathsf{cert_r}(C) = (a_1, \ldots, a_{k-1}, ?, ?, \ldots, ?) \qquad (4.17)$$

is called the *row order parent* of B. ◇

Theorem 4.2.3. *Let $B \in N_{\mathcal{R}}$ be row order compatible with row order parent $C \in N_{\mathcal{R}}$. If $\mathsf{canon_r}(B) = B$ then also $\mathsf{canon_r}(C) = C$.*

Proof : Take $\mathsf{cert_r}(B)$ and $\mathsf{cert_r}(C)$ as in (4.16) and (4.17). Suppose that $\mathsf{canon_r}(C) \neq C$. Then there must exist a $\pi \in \mathrm{Sym}(n)$ such that $\pi C <_r C$, or equivalently, $\mathsf{cert_r}(\pi C) < \mathsf{cert_r}(C)$. Assume that $l \in \{1, \dots, (n^2 - n)/2\}$ is the index of the first entry in which $\mathsf{cert_r}(\pi C)$ and $\mathsf{cert_r}(C)$ differ, and then

$$\mathsf{cert_r}(\pi C)_l < \mathsf{cert_r}(C)_l.$$

By (4.17), any undefined value ? appears in the least significant entries of $\mathsf{cert_r}(C)$. Moreover, since ? is the maximal element of Σ, $\mathsf{cert_r}(\pi C)_l \neq ?$ and therefore $l < k$. This proves that π, when regarded as a permutation of certificate indices, permutes the set $\{1, \dots, k-1\}$ and hence the $k-1$-prefix of $\mathsf{cert_r}(\pi B)$ and $\mathsf{cert_r}(\pi C)$ as the same. Hence

$$\mathsf{cert_r}(\pi B)_l = \mathsf{cert_r}(\pi C)_l < \mathsf{cert_r}(C)_l = \mathsf{cert_r}(B)_l,$$

a contradiction to $\mathsf{canon_r}(B) = B$. ■

Corollary 4.2.1. *Let $k \in \{2, \dots, (n^2 - n)/2\}$ and let $B_j \in N_{\mathcal{R}}$ be row order compatible with row order parent $B_{j+1} \in N_{\mathcal{R}}$ for each $j \in \{1, \dots, k\}$. If $\mathsf{canon_r}(B_1) = B_1$ then also $\mathsf{canon_r}(B_j) = B_j$ for each such j.* ◇

The previous theorem provides a natural way to structure the search. Recall that the exhaustive generation algorithm starts with a matrix $M \in \mathcal{M}_{n,d}$ where all non-diagonal entries are still uninstantiated. The control strategy applied during exhaustive generation is defined in such a way that the upperdiagonal matrix entries of M are systematically instantiated according to the following instantiation order:

$$\begin{array}{l} M_{1,2}, \dots, M_{1,n}, M_{2,3}, \dots, M_{2,n}, M_{3,4}, \dots \\ \quad \dots, M_{n-3,n}, M_{n-2,n-1}, M_{n-2,n}, M_{n-1,n} \end{array} \tag{4.18}$$

It is easy to see that this row by row instantiation order naturally coincides with the order in which row order certificates of matrices in $\mathcal{M}_{n,d}$ are constructed. Therefore (4.17) and (4.16) hold for every $B, C \in N_{\mathcal{R}}$ such that B is row order compatible with row order parent C. Hence, subject to the above control strategy and the canonical representative map $\mathsf{canon_r}$, the orderly generation algorithm generates exactly one representative from each orbit of $\mathrm{Sym}(n) \backslash\!\backslash \mathcal{X}_{L_0 \dots L_d}$.

Furthermore, if we instantiate each matrix entry of M by systematically assigning – in increasing order – all values of its domain to it in turn, then it is guaranteed that row order canonical representatives of matrices in $N_{\mathcal{R}}$ turn up first during the traversal of the search tree.

Note that it is not required to check the row order canonicity at each level in the search tree, by Corollary 4.2.1. Of course, it is still necessary to test the row order canonicity at the deepest level $(n^2 - n)/2$ in the recursion tree. Typically we shall only perform a canonicity test each time another row of M becomes completely instantiated. Applying this canonicity test strategy allows us to deviate from the fixed column by column instantiation order of (4.18). Whenever a domain of an uninstantiated matrix entry has size 1, we may instantiated this forced matrix entry first.

4.2.2 Column order canonical form

The column canonical form which we introduce in this section is very similar to the row canonical form from the previous section but is less frequently encountered in the literature. Let $A \in \mathcal{M}_{n,d}$, then we define the *column order certificate* $\mathsf{cert_c}(A)$ of A to be the tuple over $\Sigma \setminus \{0\}$ of length $(n^2 - n)/2$ obtained by concatenating the upperdiagonal entries of A in a *column-by-column* order ,

$$\begin{aligned} \mathsf{cert_c}(A) \quad = \quad & (A_{1,2}, A_{1,3}, A_{2,3}, A_{1,4}, \dots, A_{3,4}, \dots \\ & \dots, A_{1,n-1}, \dots, A_{n-2,n-1}, A_{1,n}, \dots, A_{n-1,n}). \end{aligned} \tag{4.19}$$

We can again use the lexicographic ordering on tuples to define a total ordering on the set $\mathcal{M}_{n,d}$. Let $A, B \in \mathcal{M}_{n,d}$ then we define $A <_c B$ if and only if $\mathsf{cert_c}(A) < \mathsf{cert_c}(B)$ and

$$\begin{aligned} \mathsf{canon_c}(A) &= \min_c (\mathrm{Sym}(n)\, A) \\ &= \min_c (\{\pi\, A \;:\; \pi \in \mathrm{Sym}(n)\}) \end{aligned} \tag{4.20}$$

The matrix $\mathsf{canon_c}(A)$ is defined to be *minimal* in its orbit and is called the *column order canonical representative* of A. If $A = \mathsf{canon_c}(A)$, we say that A is in *column order canonical form*.

Example 4.5. Consider the matrices $A, B \in \mathcal{M}_{5,2}$ as shown below.

$$A = \begin{pmatrix} 0 & \mathbf{2} & 1 & \mathbf{1} & 2 \\ 2 & 0 & 2 & \mathbf{1} & 1 \\ 1 & 2 & 0 & \mathbf{2} & 1 \\ 1 & 1 & 2 & 0 & ? \\ 2 & 1 & 1 & ? & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & \mathbf{2} & 1 & \mathbf{1} & 2 \\ 2 & 0 & 2 & \mathbf{1} & 2 \\ 1 & 2 & 0 & \mathbf{2} & 1 \\ 1 & 1 & 2 & 0 & ? \\ 2 & 2 & 1 & ? & 0 \end{pmatrix}$$

The corresponding column order certificates are

$$\begin{aligned} \mathsf{cert_c}(A) &= (\mathbf{2}, 1, 2, \mathbf{1}, \mathbf{1}, \mathbf{2}, 2, 1, 1, ?) \\ \mathsf{cert_c}(B) &= (\mathbf{2}, 1, 2, \mathbf{1}, \mathbf{1}, \mathbf{2}, 2, 2, 1, ?) \end{aligned}$$

and therefore $A <_c B$. •

In terms of the canonical representative map $\mathsf{canon_c}$, the set of column order canonical representatives is defined by

$$\mathsf{rep_c}(\mathcal{X}_{L_0 \dots L_d}) = \left\{ \Omega \in \mathcal{X}_{L_0 \dots L_d} \ : \ M_\Omega = \mathsf{canon_c}(M_\Omega) \right\}. \tag{4.21}$$

Similar as for the row order case, in order to generate $\mathsf{rep_c}(\mathcal{X}_{L_0 \dots L_d})$, the control strategy of the orderly exhaustive generation should be defined in such a way that

$$\mathsf{canon_c}(A) \in N_{\mathcal{R}} \tag{4.22}$$

for every $A \in N_{\mathcal{R}}$ and

$$\mathsf{canon_c}(C) = C \text{ if } \mathsf{canon_c}(B) = B \tag{4.23}$$

for every $B, C \in N_{\mathcal{R}}$ with C the parent of the nonroot node B.

Definition 4.2.4. The node $B \in N_{\mathcal{R}}$ shall be called *column order compatible* if and only if $\mathsf{cert_c}(B)$ is of the form

$$\mathsf{cert_c}(B) = (a_1, \dots, a_k, ?, \dots, ?) \tag{4.24}$$

for some $k \in \{1, \dots, (n^2 - n)/2\}$ and $a_i \in \{1, \dots, d\}$ for all $i \in \{1, \dots, k\}$. B is called a *root node* if $k = 0$ in the above. If B is not a root node, then the unique matrix $C \in N_{\mathcal{R}}$ such that

$$\mathsf{cert_c}(C) = (a_1, \dots, a_{k-1}, ?, ?, \dots, ?) \tag{4.25}$$

is called the *column order parent* of B. ◇

Theorem 4.2.5. *Let $B \in N_{\mathcal{R}}$ be column order compatible with column order parent $C \in N_{\mathcal{R}}$. If $\mathsf{canon_c}(B) = B$ then also $\mathsf{canon_c}(C) = C$.*

Proof : The column order case is proved in a similar way as in Theorem 4.2.3. ■

Corollary 4.2.2. *Let $k \in \{2, \ldots, (n^2 - n)/2\}$ and let $B_j \in N_{\mathcal{R}}$ be column order compatible with column order parent $B_{j+1} \in N_{\mathcal{R}}$ for each $j \in \{1, \ldots, k\}$. If $\mathsf{canon_c}(B_1) = B_1$ then also $\mathsf{canon_c}(B_j) = B_j$ for each such j.* ◇

The previous theorem provides a natural way to structure the search. Similar as in the row order case, we define the generation algorithm's control strategy in such a way that the instantiation order naturally coincides with the order in which column order certificates of matrices in $\mathcal{M}_{n,d}$ are constructed. More precisely, upperdiagonal matrix entries of M are systematically instantiated according to the following instantiation order:

$$\begin{array}{l} M_{1,2}, M_{1,3}, M_{2,3}, M_{1,4}, \ldots, M_{3,4}, M_{1,5}, \ldots \\ \quad \ldots, M_{n-3,n-2}, M_{1,n-1}, \ldots, M_{n-2,n-1}, M_{1,n}, \ldots, M_{n-1,n} \end{array} \tag{4.26}$$

Therefore (4.25) and (4.24) hold for every $B, C \in N_{\mathcal{R}}$ such that B is column order compatible with column order parent C and hence the orderly generation algorithm shall generate exactly one representative from each orbit of $\mathrm{Sym}(n) \setminus\!\!\setminus \mathcal{X}_{L_0 \ldots L_d}$. Furthermore, if we instantiate each matrix entry of M by systematically assigning – in increasing order – all values of its domain to it in turn, then again it is guaranteed that column order canonical representatives of matrices in $N_{\mathcal{R}}$ turn up first during the traversal of the search tree.

Checking column order canonicity at each level in the search tree is not required, by Corollary 4.2.2. We shall only perform a canonicity test each time another upperdiagonal column of M becomes completely instantiated. At that point, it suffices to check the canonicity of the fully instantiated leading principal submatrix of M. Again, whenever a domain of an uninstantiated matrix entry has size 1, we may instantiated this forced matrix entry first.

4.2.3 Lexically ordered matrices

Definition 4.2.6. A matrix $A \in \mathcal{M}_{n,d}$ is called *lexically ordered* if and only if

$$(A_{i,1}, \ldots, A_{i,n}) \leq (A_{i+1,1}, \ldots, A_{i+1,n})$$

for every $i \in \{1, \ldots, n-1\}$. $\diamond$

Theorem 4.2.7. *Let $A \in \mathcal{M}_{n,d}$. If $A = \mathsf{canon_r}(A)$, then A is lexically ordered. Similarly, if $A = \mathsf{canon_c}(A)$, then A is lexically ordered.*

Proof : Assume that A is not lexically ordered. Let i be the smallest row number for which the $i+1$-th row of A is lexically smaller than the i-th row of A. Let j be the smallest column number for which those rows differ. More precisely we have

$$A_{i,j} > A_{i+1,j} \quad \text{and} \quad A_{i,k} = A_{i+1,k}$$

for each $k \in \{1, \ldots, j-1\}$, and hence by symmetry,

$$A_{j,i} > A_{j,i+1} \quad \text{and} \quad A_{k,i} = A_{k,i+1}.$$

Because A has zero diagonal, we must have $j < i$. But then it is easily seen that the transposition $\sigma = (i\; i+1)$ which interchanges the i-th and $i+1$-th row (and column) maps A onto a matrix σA such that $\mathsf{cert_r}(\sigma A) < \mathsf{cert_r}(A)$. Hence $A \neq \mathsf{canon_r}(A)$. The column order case is proved in exactly the same way. ∎

The following example illustrates Theorem 4.2.7.

Example 4.6. Let $\sigma = (3\ 4) \in \mathrm{Sym}(5)$. Consider the matrix $A \in \mathcal{M}_{5,2}$ as shown below.

$$A = \begin{pmatrix} 0 & 2 & 1 & 1 & 2 \\ 2 & 0 & 2 & 1 & 1 \\ \mathbf{1} & \mathbf{2} & \mathbf{0} & \mathbf{2} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{0} & \mathbf{?} \\ 2 & 1 & 1 & ? & 0 \end{pmatrix} \qquad \sigma A = \begin{pmatrix} 0 & 2 & 1 & 1 & 2 \\ 2 & 0 & 1 & 2 & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{2} & \mathbf{?} \\ \mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{0} & \mathbf{1} \\ 2 & 2 & ? & 1 & 0 \end{pmatrix}$$

The matrix A is not lexically ordered since the fourth row of A is lexically smaller than the third row of A. Note that $\sigma A <_r A$ and $\sigma A <_c A$, hence A is not in row or column order canonical form.

•

Checking whether a matrix of $\mathcal{M}_{n,d}$ is in row or column order canonical form is a computational demanding task. Lexical ordering is a necessary condition for a matrix of $\mathcal{M}_{n,d}$ being in row or column order canonical form. From an algorithmic point of view, this necessary condition turns out to be straightforward and easy to verify. That is why we opt to check this necessary condition at each level in the recursion tree. An even stronger necessary condition for the minimality of the matrices of $\mathcal{M}_{n,d}$ can be establish as follows.

Definition 4.2.8. A matrix $A \in \mathcal{M}_{n,d}$ is called 0-*lexically ordered* if and only

$$(A_{i,1}, \ldots, A_{i,i-1}, A_{i,i+2}, \ldots, A_{i,n}) \leq \\ (A_{i+1,1}, \ldots, A_{i+1,i-1}, A_{i+1,i+2}, \ldots, A_{i+1,n})$$

for every $i \in \{1, \ldots, n-1\}$. ◇

Corollary 4.2.3. *Let $A \in \mathcal{M}_{n,d}$. If A is 0-lexically ordered, then A is also lexically ordered.* ◇

Theorem 4.2.9 establishes a connection between minimality and 0-lexically ordering for row and column order representatives.

Theorem 4.2.9. *Let $A \in \mathcal{M}_{n,d}$. If $A = \mathsf{canon}_\mathsf{r}(A)$, then A is 0-lexically ordered. Similarly, if $A = \mathsf{canon}_\mathsf{c}(A)$, then A is 0-lexically ordered.*

Proof : Assume that A is not 0-lexically ordered. Let i be the smallest row number for which

$$(A_{i,1}, \ldots, A_{i,i-1}, A_{i,i+2}, \ldots, A_{i,n}) > \\ (A_{i+1,1}, \ldots, A_{i+1,i-1}, A_{i+1,i+2}, \ldots, A_{i+1,n})$$

Let j be the smallest column number for which those tuples differ. If $j < i$ then according to Theorem 4.2.7 we find that $A \neq \mathsf{canon}_\mathsf{r}(A)$. Otherwise, there must exist a $k \in \{i+2, \ldots, n\}$ such that

$$A_{i,k} > A_{i+1,k} \qquad A_{i,l} = A_{i+1,l} \qquad A_{i,p} = A_{i+1,p}$$

for each $l \in \{1, \ldots, i-1\}$ and each $p \in \{i+2, \ldots, k-1\}$, and hence by symmetry,

$$A_{k,i} > A_{k,i+1} \qquad A_{l,i} = A_{l,i+1} \qquad A_{p,i} = A_{p,i+1}.$$

Because A is symmetric and has zero diagonal, it is easily seen that the transposition $\sigma = (i\ i+1)$ which interchanges the i-th and $i+1$-th row (and column) maps A onto a matrix σA such that $\mathsf{cert_r}(\sigma A) < \mathsf{cert_r}(A)$. Hence $A \neq \mathsf{canon_r}(A)$. The column order case is proved in exactly the same way. ■

The following example illustrates Theorem 4.2.9.

Example 4.7. Let $\sigma = (3\ 4) \in \mathrm{Sym}(5)$. Consider the matrix $A \in \mathcal{M}_{5,2}$ as shown below.

$$A = \begin{pmatrix} 0 & 2 & 1 & 1 & 2 \\ 2 & 0 & 2 & 2 & 1 \\ \mathbf{1} & \mathbf{2} & 0 & 2 & \mathbf{2} \\ \mathbf{1} & \mathbf{2} & 2 & 0 & \mathbf{1} \\ 2 & 1 & 2 & 1 & 0 \end{pmatrix} \qquad \sigma A = \begin{pmatrix} 0 & 2 & 1 & 1 & 2 \\ 2 & 0 & 2 & 2 & 1 \\ \mathbf{1} & \mathbf{2} & 0 & 2 & \mathbf{1} \\ \mathbf{1} & \mathbf{2} & 2 & 0 & \mathbf{2} \\ 2 & 2 & 1 & 2 & 0 \end{pmatrix}$$

The matrix A is not 0-lexically ordered. Indeed the tuple $(A_{4,1}, A_{4,2}, A_{4,5})$ is lexically smaller than the tuple $(A_{3,1}, A_{3,2}, A_{3,5})$. Note that $\sigma A <_r A$ and $\sigma A <_c A$, hence A is not in row or column order canonical form. •

4.3 Checking the algebraic properties of the minimal idempotents

In Section 4.1.4 we described in general how we can prune the recursion tree based on the positive semidefiniteness and the rank of the minimal idempotents E_i of the d-class association schemes in $\mathcal{X}_{L_0 \ldots L_d}$. We could indeed use the strategy to check for every partially instantiated matrix $M \in N_{\mathcal{R}}$ we encounter during traversal, whether any fully instantiated principal submatrix of the related matrices M_{E_i} is not positive semidefinite and has a rank more than Q_{0i}, where Q denotes the dual eigenmatrix of the association schemes in $\mathcal{X}_{L_0 \ldots L_d}$. However, during the actual generation algorithm we shall use a somewhat more relaxed version of this strategy. Define $\mathsf{lead}(M_{E_i})$ to be the largest leading principal submatrix of M_{E_i} which is fully instantiated. Then roughly, we will only check the positive semidefiniteness and the rank of $\mathsf{lead}(M_{E_i})$, instead of checking these algebraic constraints for all principal submatrices of M_{E_i}.

4.3.1 Positive semidefiniteness algorithm

Recall from linear algebra that a real symmetric matrix $A \in \mathbf{R}^{v \times v}$ is *positive semidefinite* if and only if

$$x\,A\,x^T \geq 0 \tag{4.27}$$

for every row vector $x \in \mathbf{R}^{1 \times v}$ and its transpose $x^T \in \mathbf{R}^{v \times 1}$. If the matrix A is *not* positive semidefinite then we define a *witness* for A to be any row vector x such that $x\,A\,x^T < 0$.

Theorem 4.3.1. *Consider a real symmetric matrix $A \in \mathbf{R}^{v \times v}$, where*

$$A = \begin{pmatrix} \alpha & a \\ a^T & A' \end{pmatrix} \tag{4.28}$$

with $\alpha \in \mathbf{R}$, $a \in \mathbf{R}^{1 \times v-1}$ and $A' \in \mathbf{R}^{v-1 \times v-1}$. Then we distinguish between the following cases:

1. *If $\alpha < 0$, then A is not positive semidefinite. Moreover, $x = (1\ 0\ \ldots\ 0) \in \mathbf{R}^{1 \times v}$ is a witness for A.*
2. *If $\alpha > 0$, then A is positive semidefinite if and only if*
$$A' - \frac{a^T a}{\alpha} \tag{4.29}$$
is positive semidefinite. If $y \in \mathbf{R}^{1 \times v-1}$ is a witness for $A' - \frac{a^T a}{\alpha}$ then
$$x = (-\frac{y a^T}{\alpha}\ y)$$
is a witness for A.
3. *If $\alpha = 0$, then A is positive semidefinite if and only if A' is positive semidefinite and $a = 0$. If $a \neq 0$, then we may find $y \in \mathbf{R}^{1 \times v-1}$ such that $y a^T \geq 0$ and then each vector $x = (\lambda\ y)$ is a witness for A whenever*
$$\lambda < \frac{-y A' y^T}{2 y a^T}.$$
If A' is not positive semidefinite, then every witness y for A' can be extended to a witness $x = (0\ y)$ for A.

Proof : Let $\lambda \in \mathbf{R}$, $y \in \mathbf{R}^{1\times v-1}$ and set $x = (\lambda\ y)$. We have

$$\begin{aligned} xAx^T &= (\lambda \quad y) \begin{pmatrix} \alpha & a \\ a^T & A' \end{pmatrix} \begin{pmatrix} \lambda \\ y \end{pmatrix} \\ &= \lambda^2\alpha + 2\lambda y a^T + yA'y^T. \end{aligned} \tag{4.30}$$

We consider the following three different cases:

1. If $\alpha < 0$, then the right hand side of (4.30) is less than zero for $\lambda \in \mathbf{R}^+$ and $y = 0$. Hence we find that the matrix A is not positive semidefinite with witness $(1\ 0\ \ldots\ 0)$.

2. If $\alpha > 0$, then we may rewrite the right hand side of (4.30) as

$$xAx^T = \alpha\left(\lambda + \frac{ya^T}{\alpha}\right)^2 + y\left(A' - \frac{a^Ta}{\alpha}\right)y^T, \tag{4.31}$$

   using $ya^T = ay^T$. This expression is positive or zero for every x if and only if

$$y\left(A' - \frac{a^Ta}{\alpha}\right)y^T \geq 0$$

   for every y, i.e., if and only if the matrix $A' - \frac{a^Ta}{\alpha} \in \mathbf{R}^{v-1\times v-1}$ is positive semidefinite. If this matrix is not positive semidefinite, and y is a witness for the matrix, then for

$$\lambda = -\frac{ya^T}{\alpha}$$

   the vector $(\lambda\ y)$ provides a witness for A.

3. Finally if $\alpha = 0$, then the right hand side of (4.30) reduces to

$$xAx^T = 2\lambda ya^T + yA'y^T, \tag{4.32}$$

   which is linear in λ. This expression is positive for all λ if and only if the coefficient $2ya^T$ of λ is zero and the constant term $yA'y^T$ is positive. Hence the matrix A is positive semidefinite if and only if $ya^T = 0$ and $yA'y^T \geq 0$ for all y, or equivalently, if and only if $a = 0$ and the matrix A' is positive semidefinite.

As a consequence, if A' is not positive semidefinite and y is a witness for A', then the vector $(0\ y)$ provides a witness for A. Also, if $a \neq 0$, then we may find y such that $ya^T > 0$ and then any λ satisfying

$$\lambda < -\frac{yA'y^T}{2ya^T}$$

will make (4.32) less than zero.

■

This theorem can easily be used as the basis for an algorithm which checks whether a given real symmetric matrix $A \in \mathbf{R}^{v\times v}$ is positive semidefinite. To simplify notations, we denote the element on the i-th row and j-th column of A by $A[i,j]$ (and not by $A_{i,j}$ as customary in other circumstances). Submatrices (or matrices derived from submatrices) keep the row and column numbering of the matrices they are part of. For example, the rows and columns of the matrices A and A' in Theorem 4.3.1 would be numbered from 1 up to v and from 2 up to v respectively.

Using the notations of Theorem 4.3.1, we define

$$A^{(2)} = \begin{cases} A', & \text{when } \alpha = 0, \\ \\ A' - \dfrac{a^Ta}{\alpha}, & \text{when } \alpha \neq 0. \end{cases} \tag{4.33}$$

The matrix obtained by applying the same process to $A^{(2)}$ shall be denoted by $A^{(3)}$, and in a similar way we may define $A^{(4)}, A^{(5)}, \ldots, A^{(v)}$. We also write $A^{(1)} = A$.

In general, the matrix $A^{(k)}$ is a symmetric $(v-k+1)\times(v-k+1)$ matrix with rows and columns numbered from k up to v, by the numbering conventions introduced earlier. This yields the following recurrence relation, for all $i, j \geq k+1$:

$$A^{(k+1)}[i,j] = \begin{cases} A^{(k)}[i,j], & \text{when } A^{(k)}[k,k] = 0, \\ \\ A^{(k)}[i,j] - \dfrac{A^{(k)}[i,k]A^{(k)}[k,j]}{A^{(k)}[k,k]}, & \text{when } A^{(k)}[k,k] \neq 0. \end{cases} \tag{4.34}$$

Theorem 4.3.1 leads to Algorithm 4.1 which takes a real symmetric $v \times v$ matrix A as input and returns **true** or **false** depending on whether A is positive semidefinite or not. It is easily seen that Algorithm 4.1 needs $O(v^3)$ operations in the worst case. Storage requirements are only $O(v^2)$ because $A^{(k+1)}[i,j]$ can be stored in the same place as $A^{(k)}[i,j]$. Also note that every $A^{(k)}$ is symmetric and therefore only about half of each matrix needs to be stored.

Algorithm 4.1 Checks whether a real symmetric $v \times v$ matrix A is positive semidefinite or not.

```
function isPSD(A : matrix) : boolean
1: $k \leftarrow 1$
2: while $k < v$ do
3:   if $A^{(k)}[k,k] < 0$ then
4:     return false
5:   else if $A^{(k)}[k,k] = 0$ then
6:     for $j \leftarrow k+1 \dots v$ do
7:       if $A^{(k)}[j,k] \neq 0$ then
8:         return false
9:     compute $A^{(k+1)}$ according to (4.34)
10:    $k \leftarrow k+1$
11:  else
12:    compute $A^{(k+1)}$ according to (4.34)
13:    $k \leftarrow k+1$
14: return $A^{(v)}[v,v] \geq 0$
```

The next version of the positive semidefiniteness algorithm stems from the observation that all comparisons in Algorithm 4.1 are performed on elements $A^{(k)}[i,j]$ with either $k = i$ or $k = j$. Define $B \in \mathbf{R}^{v \times v}$ with entries $B[i,j] = A^{(i)}[i,j]$. Note that $B[i,j]$ is only defined when $i \leq j$ and that $B[1,j] = A[1,j]$.

We may now reformulate (4.34) as follows, for all $i, j \geq k+1$:

$$A^{(k+1)}[i,j] = \begin{cases} A^{(k)}[i,j], & \text{when } A^{(k)}[k,k] = 0, \\ A^{(k)}[i,j] - \dfrac{B[k,i]B[k,j]}{B[k,k]}, & \text{when } A^{(k)}[k,k] \neq 0. \end{cases}$$

and hence, by repeated application for different k

$$\begin{aligned} A^{(k+1)}[i,j] &= A^{(1)}[i,j] - \frac{B[1,i]B[1,j]}{B[1,1]} - \frac{B[2,i]B[2,j]}{B[2,2]} - \cdots \\ &\quad \cdots - \frac{B[k-1,i]B[k-1,j]}{B[k-1,k-1]} - \frac{B[k,i]B[k,j]}{B[k,k]}, \end{aligned} \tag{4.35}$$

where all fractions with zero denominator $B[j,j]$ must be regarded as equal to zero. From (4.35) we obtain a recurrence relation for B :

$$\begin{aligned} B[i,j] &= A[i,j] - \frac{B[1,i]B[1,j]}{B[1,1]} - \frac{B[2,i]B[2,j]}{B[2,2]} - \cdots \\ &\quad \cdots - \frac{B[i-2,i]B[i-2,j]}{B[i-2,i-2]} - \frac{B[i-1,i]B[i-1,j]}{B[i-1,i-1]}, \end{aligned} \tag{4.36}$$

again omitting all terms with a zero denominator.

In terms of (4.36), Theorem 4.3.1 leads to Algorithm 4.2 which again takes a real symmetric $v \times v$ matrix A as input and returns **true** or **false** depending on whether A is positive semidefinite or not. Algorithm 4.2 again needs $O(v^3)$ operations and $O(v^2)$ storage.

As stated before, during traversal of the recursion tree, we shall typically only check the positive semidefiniteness of leading principal submatrices $\mathsf{lead}(M_{E_i})$. Moreover, consider two matrices M and $M' \in N_{\mathcal{R}}$ such that M' is the parent of M in the recursion tree, then obviously we only have to check whether $\mathsf{lead}(M_{E_i})$ is positive semidefinite if $\mathsf{lead}(M'_{E_i}) \neq \mathsf{lead}(M_{E_i})$. If so, a straightforward way to check $\mathsf{lead}(M_{E_i})$ would be to apply Algorithm 4.2 with $\mathsf{lead}(M_{E_i})$ as input. However, during the course of the generation algorithm we will frequently have to check whether a given leading principal submatrix is positive semidefinite. This repeated application will allow us to improve Algorithm 4.2.

According to (4.36), the value of $B[i,j]$ with $i \leq j$ depends only on the value of $A[i,j]$ and the values of $B[k,l]$ such that $k < i$ and $l \leq j$. Hence by repeated application of (4.36), we find that the value of $B[i,j]$ can be expressed entirely in terms of entries $A[k,l]$ such that $k \leq i$ and $l \leq j$. Given two matrices A and A' such that $A[k,l] \neq A'[k,l]$ only when either $k > i$ or $l > j$, then if we subsequently apply Algorithm 4.2 to first A and then A', we can reuse the value of $B[i,j] = B'[i,j]$ (and a fortiori, all values of $B[x,y] = B'[x,y]$ with $x \leq i$ and $y \leq j$) when applying Algorithm 4.2 to A'.

Algorithm 4.2 Checks whether a real symmetric $v \times v$ matrix A is positive semidefinite or not.

```
function isPSD(A : matrix) : boolean
for $j \leftarrow 1 \ldots v$ do
    $B[1,i] \leftarrow A[1,i]$
$k \leftarrow 1$
while $k < v$ do
    if $B[k,k] < 0$ then                                    ❶
        return false
    else if $B[k,k] = 0$ then
        for $j \leftarrow k+1 \ldots v$ do
            if $B[k,j] \neq 0$ then
                return false
        $k \leftarrow k+1$
    else
        for $j \leftarrow k+1 \ldots v$ do
            compute $B[k+1,j]$ according to (4.36)
        $k \leftarrow k+1$
return $B[v,v] \geq 0$
```

Furthermore, assume that A and A' are subsequentially checked by Algorithm 4.2 during traversal, then at least $A[x, y] = A'[x, y]$ for every $x < q$ and $y < q$ where q denotes the order of A'. Hence, in the worst case Algorithm 4.2 should only compute the values $B'[k, q]$ such that $k \leq q$, when checking the positive semidefiniteness of A'.

4.3.2 Preemptive positive semidefiniteness checking

Sometimes it is possible to decide that A is not positive semidefinite before the leading principal submatrix is completely instantiated. One of the decision criteria to abort the iteration in Algorithm 4.2, is when the value of $B[k, k] < 0$ for some $k \in \{1, \ldots, v\}$ (cf. ❶ in Algorithm 4.2). From (4.36) we obtain a recurrence relation for $B[k, k]$:

$$\begin{aligned} B[k,k] &= A[k,k] - \frac{B[1,k]^2}{B[1,1]} - \frac{B[2,k]^2}{B[2,2]} - \cdots \\ &\quad \cdots - \frac{B[k-2,k]^2}{B[k-2,k-2]} - \frac{B[k-1,k]^2}{B[k-1,k-1]}, \end{aligned} \tag{4.37}$$

again omitting all terms with a zero denominator. Note that when we check the value of $B[k, k]$ in Algorithm 4.2, all denominators in (4.37) are necessarily positive or zero. Suppose that we instantiate the matrix entry $A[i, k]$ for some $i < k$. If all $A[x, y]$ such that $x \leq i$ and $y \leq k$ are instantiated, then we can already compute the value of

$$\begin{aligned} B_i[k,k] &= A[k,k] - \frac{B[1,k]^2}{B[1,1]} - \frac{B[2,k]^2}{B[2,2]} - \cdots \\ &\quad \cdots - \frac{B[i-2,k]^2}{B[i-2,i-2]} - \frac{B[i-1,k]^2}{B[i-1,i-1]} \end{aligned}$$

If $B_i[k, k] < 0$, then we can decide that A is not positive semidefinite, as $B_i[k, k] \geq B[k, k]$.

Remark 4.1. Prematurely checking positive semidefiniteness allows us to prune our generation process substantially higher in the recursion tree. Especially when using a row by row instantiation order, this technique turns out to be effective. After all, in contrast to the column by column approach, large leading principal submatrices which are fully instantiated are not immediately obtained. •

4.3.3 Elimination of numerical errors

In order to check positive semidefiniteness, we need to perform a lot of floating point computations, which by their nature only yield approximate results. It is important to be absolutely certain that these numerical errors are sufficiently small as not to invalidate classification results obtained by our generation algorithm.

To ensure correctness, Algorithm 4.2 is modified in such a way that before **false** is returned, a witness x for A is computed according to Theorem 4.3.1. Afterwards, to catch accumulations of rounding errors, the expression xAx^T is re-evaluated and compared to $-\epsilon$ for a suitably small $\epsilon > 0$. Rounding errors on the value of xAx^T can be estimated very accurately. If the witness x fails this test, then we opt not to prune the recursion tree. Of course this may mean that we sometimes falsely assume that a matrix is positive semidefinite when it is not. This, however, is not a problem.

4.3.4 Look-back strategy

The witness x constructed when A is not positive semidefinite is not only used to eliminate numerical errors but also allows us to identify a smaller principal submatrix A'' of A which is also not positive semidefinite. Indeed, assume that only for the elements $l \in \{j_1, \ldots, j_w\} \subseteq \{1, \ldots, v\}$ with $j_i < j_k$ if $i < k$ the corresponding witness entry $x_l \neq 0$. Then the principal submatrix A'' of A which is indexed by the elements of $\{j_1, \ldots, j_w\}$ is also not positive semidefinite. After all, the vector $y \in \mathbf{R}^{1 \times w}$ such that $y_p = x_{j_p}$ for every $p \in \{1, \ldots, w\}$ is a witness for A'', as $yA''y^T < 0$.

This leads to the following look-back strategy which we can incorporate into our generation algorithm. Suppose that the instantiation of a matrix entry A completes the submatrix A'' with the properties above, then this instantiation remains forbidden until a backtrack occurs to the second last instantiated matrix entry of A which contributed to the submatrix A''.

A wiutness for A needs not necessarily be constructed according to Theorem 4.3.1, indeed any other vector $z \in \mathbf{R}^{1\times v}$ such that $zAz^T < 0$ can be used as a witness for A. Theorem 4.3.2, a slight improvement to this theorem, provides us with a witness z for which many of the entries are zero. Using this alternative, we usually can identify smaller principal submatrices A'' of A which are not positive semidefinite. This allows us to further improve the look-back strategy outlined above.

Theorem 4.3.2. *Consider a real symmetric matrix $A \in \mathbf{R}^{v\times v}$, where*

$$A = \begin{pmatrix} \alpha & a \\ a^T & A' \end{pmatrix}$$

with $\alpha \in \mathbf{R}$, $a \in \mathbf{R}^{1\times v-1}$ and $A' \in \mathbf{R}^{v-1\times v-1}$. Then we distinguish between the following cases:

1. *If $\alpha < 0$, then A is not positive semidefinite. Moreover, $z = (1\ 0\ \ldots\ 0) \in \mathbf{R}^{1\times v}$ is a witness for A.*
2. *If $\alpha > 0$, then A is positive semidefinite if and only if $A' - \frac{a^Ta}{\alpha}$ is positive semidefinite. If $y \in \mathbf{R}^{1\times v-1}$ is a witness for $A' - \frac{a^Ta}{\alpha}$, then $z = (\lambda\, y)$ is a witness for A, with*

$$\lambda = \begin{cases} 0 & \text{if } \frac{\left(ya^T\right)^2}{\alpha} + y\left(A' - \frac{a^Ta}{\alpha}\right)y^T < 0 \\ -\frac{ya^T}{\alpha} & \text{otherwise} \end{cases} \tag{4.38}$$

3. *If $\alpha = 0$, then A is positive semidefinite if and only if A' is positive semidefinite and the vector $a = 0$. If $a \neq 0$, the we may find $y \in \mathbf{R}^{1\times v-1}$ such that $ya^T \geq 0$ and then each vector $z = (\lambda\ y)$ is a witness for A whenever $\lambda < \frac{-yA'y^T}{2ya^T}$. If A' is not positive semidefinite, then every witness y for A' can be extended to a witness $z = (0\ y)$ for A.*

Proof : The case where $\alpha < 0$ or $\alpha = 0$ is equivalent with Theorem 4.3.1. If $\alpha > 0$, then only the witness is constructed differently. Setting $\lambda = 0$ in (4.31) yields condition (4.38). ∎

4.3.5 Checking the rank constraint

In the more relaxed version of the rank constraint, we shall typically only check the rank of the leading principal submatrices $\mathsf{lead}(M_{E_i})$. Moreover, consider two matrices M and $M' \in N_{\mathcal{R}}$ such that M' is the parent of M in the recursion tree. Obviously the rank of $\mathsf{lead}(M_{E_i})$ needs only to be checked if the order of $\mathsf{lead}(M_{E_i})$ is larger than the rank Q_{0i} of the minimal idempotent E_i and $\mathsf{lead}(M'_{E_i}) \neq \mathsf{lead}(M_{E_i})$. Finally, note that if $\mathsf{lead}(M_{E_i})$ is of order k and has rank q such that $q = Q_{0i} - p$, then for all partially instantiated matrices $M'' \in N_{\mathcal{R}}$ which are part of the subtree rooted at M, it is pointless to check the rank of the associated matrix $\mathsf{lead}(M''_{E_i})$ unless its order is larger than $k + p + 1$.

A straightforward manner to compute the rank of $\mathsf{lead}(M_{E_i})$ is to use standard Gaussian elimination. However, some of the entries of $\mathsf{lead}(M_{E_i})$ may be non-integral. Since using floating point computations for Gaussian elimination only yields approximate results, this method is almost useless for our purposes. Recall from linear algebra that the rank of a matrix A with integral entries, is always greater than or equal to the rank of the matrix $\bar{A}$ obtained by reducing every matrix entry of A modulo a fixed prime number p. Indeed, if the i-th and j-th row of A are linearly dependent, then also the i-th and j-th row of $\overline{A}$ are linearly dependent. However, if the i-th and j-th row of A are linearly independent, then it is not necessarily so that the i-th and j-th row of $\overline{A}$ are also linearly independent. If we choose $p < 2^{15}$ (e.g. $p = 32749$), then computing the rank of $\overline{A}$ can be done exactly, because then integer overflow can be avoided.

Depending on the actual matrix entries, we may transform $A = \mathsf{lead}(M_{E_i})$ to an appropriate integral matrix $\overline{A}$, as follows:

Rational numbers If all matrix entries of A are rational numbers, then each entry is multiplied by the least common multiple of the denominators of these matrix entries so as to obtain an integral matrix $\overline{A}$.

Irrational numbers If some matrix entries of A are irrational numbers, then it is still possible to find a suitable p and to transform A into an appropriate integral matrix $\overline{A}$. Example 4.8 illustrates this in the case of the Perkel graph.

Finally, note that the rank of $\overline{A}$ modulo p is less than or equal to the rank of A.

Of course this may mean that if we check the rank of $\overline{A}$ modulo p, we sometimes falsely assume that A has rank less than or equal to Q_{0i} when it is not. This, however, is not a problem.

Example 4.8. The Perkel graph is the unique (up to isomorphism) distance regular graph of order 57, degree 6 and diameter 3 with intersection array $\{6, 5, 2; 1, 1, 3\}$. Recall from Example 2.13 that its dual eigenmatrix Q is given by

$$Q = \begin{pmatrix} 1 & 18 & 18 & 20 \\ 1 & \frac{9}{2} + \frac{3\sqrt{5}}{2} & \frac{9}{2} - \frac{3\sqrt{5}}{2} & -10 \\ 1 & -\frac{3}{2} + \frac{9\sqrt{5}}{10} & -\frac{3}{2} - \frac{9\sqrt{5}}{10} & 2 \\ 1 & -9\sqrt{5} & 9\sqrt{5} & -1 \end{pmatrix}$$

Consider the minimal idempotent E_2 defined by

$$E_2 = \frac{1}{57} \sum_{j=0}^{3} Q_{j2} A_j. \tag{4.39}$$

To handle the irrational element $\sqrt{5}$, we choose $p = 30011$. Then $6583^2 = 5 \mod p$ and we may first subsitute 6583 for $\sqrt{5}$ in Q_{12}, Q_{22} and Q_{32} and then multiply each coefficient of (4.39) by 570 to construct an appropriate integral matrix. •

4.4 Row versus column

When designing orderly generation algorithms we face the challenge to select an appropriate canonical form which interacts well with both the control strategy of the backtrack algorithm as well as its associated set of constraints. This requires some experimentation with different ways of structuring the search and different types of canonical forms. In Sections 4.2.1 and 4.2.2 we introduced two distinct types of canonical forms which we shall evaluate in this section.

4.4.1 Recorded objects

Typically the application of a row order canonical form requires that the the backtrack algorithm instantiates matrix entries of M in a row by row manner, whereas the application of a column order canonical form requires that entries are instantiated column by column. When using a row order or column order canonical form, we shall only perform an explicit canonicity test at intermediate levels in the recursion tree, that is, when a new row or column of M becomes completely instantiated, respectively.

Testing whether a matrix is in canonical form subject to some lexicographic order is often computational hard. The only known approach is to execute this canonicity test using backtrack search through all possible permutations of the rows and columns of M. Different pruning criteria may be used to prune the recursion tree. Nevertheless, designing an effective canonicity algorithm which turns out to be fast enough in practice is a quite complicated and subtle task.

Experimenting with both types of canonical forms would require that we design and implement a canonicity algorithm for each type. However, the effort expended in designing two different algorithms can be avoided by simulating such a canonicity test. Since the control strategy of the backtrack algorithm guarantees that row and column order representatives turn up first during traversal of the search tree, we could maintain a global set Δ of matrices encountered so far. Whenever during traversal a matrix M is considered, we then first check whether it is isomorphic with one of the recorded matrices in Δ. If M turns out to be isomorphic, then the subtree rooted at M can be pruned, after all M cannot be in canonical form.

In practice, the set Δ contains *certificates* of the matrices encountered so far – instead of the matrices themselves. These certificates are computed using `nauty` [71] and typically the set Δ is implemented using a *hash table* or a *trie* [91] which both support fast access. In order to check whether M is isomorphic with one of the recorded matrices in Δ, we first compute the certificate of M. Afterwards, we check whether this certificate is already stored in Δ. If so, we may prune the search tree rooted at M, otherwise this certificate is added to Δ.

Using this approach of recorded objects, we can evaluate how both canonical forms interact with the applied control strategy and constraints without having to implement a canonicity algorithm. Only for the canonical form with the best interaction, we shall design an effective canonicity algorithm. This canonicity algorithm is described in detail in Chapter 5.

Remark 4.2. The software package `nauty` supports the computation of a canonical labeling of a colored graph. Typically the matrices M we encounter are transformed into colored graphs and `nauty` is applied to these colored graphs so as to obtain certificates for M. For a discussion of the actual algorithm used by `nauty`, we refer the reader to [65, Section 5.6] and [71, 77]. •

Remark 4.3. Isomorph-free exhaustive generation can be obtained using the technique of recording objects. However some fundamental problems are intrinsic to this isomorphism rejection technique. First of all, the fact that we need to store each object we encounter, strongly limits the application of this approach. Indeed, when the search space is characterized by a large amount of partially instantiated matrices, storage space can rather soon turn out to be insufficient. A second problem is that it is hard to parallelize exhaustive search, since each search process must be able to access the objects recorded by other parallel search processes. •

4.4.2 Look-ahead on lexical ordering

The application of a row order as well as a column order canonical form in our generation algorithm were experimentally evaluated by generating, up to isomorphism, several strongly regular graphs with certain (v, k, λ, μ) parameter sets. The generation algorithm used for our experiments, differs only from the general generation algorithm – as previously described in this chapter – in the following points:

- Any explicit canonicity test is replaced by the isomorphism rejection technique of recording objects. Note that `nauty` is used to compute certificates.
- It is easily seen that the first two rows (and at the same time also the first two columns) of the $v \times v$ matrix M being generated are completely defined by the given (v, k, λ, μ) parameter set and the lexical ordering constaint imposed on M [20, 40]. More precisely, the first row of M is

necessarily of the form

$$\begin{array}{llll} M_{1,2} & = \ldots = & M_{1,k+1} & = 1 \\ M_{1,k+2} & = \ldots = & M_{1,v} & = 2 \end{array} \tag{4.40}$$

while the second row is necessarily of the form

$$\begin{array}{llll} M_{2,3} & = \ldots = & M_{2,2+\lambda} & = 1 \\ M_{2,3+\lambda} & = \ldots = & M_{2,k+1} & = 2 \\ M_{2,k+2} & = \ldots = & M_{1,v-\mu} & = 1 \\ M_{2,v-\mu+1} & = \ldots = & M_{2,v} & = 2 \end{array} \tag{4.41}$$

The domains of the matrix entries of the first two rows are reduced – before the actual start of the search process – in such a way that the first two rows of M are intially forced to be instantiated according to (4.40) and (4.41). Note that similar restrictions on the first two rows and columns can be deduced in the more general case of d-class association schemes.

Our experiments indicate that both the runtime as well as the size of the search tree are substantially smaller when applying a row order canonical form during generation. However, if we analyze the behaviour of the generation algorithm in detail, we observe that when we use a column order canonical form, the lexical ordering constraint imposed on M suffers heavily from trashing. This behaviour is largely due to the fact that the control strategy applied, instantiates the matrix entries of M in a column by column order.

As stated in Section 3.1, trashing behaviour in backtrack algorithms can be improved by dynamically using look-ahead techniques during the search. The explicit lexical ordering on the rows of M gives rise to the concept of a *cell structure*. This concept is central to the look-ahead techniques which we shall introduce to reduce the trashing behaviour of the lexical ordering constraint. Fix a row number r. The set of elements $j \in \{1, \ldots, v\}$ for which the vectors $(M_{1,j}, \ldots, M_{r,j})$ are equal, is called a *cell*[3] at row r of M. The set of all cells at a row r of M, is called the cell structure at row r. Due to the lexicographic ordering imposed on M, all elements which belong to the same cell correspond to adjacent columns in M. Hence cells are intervals.

Example 4.9. Consider the lexically ordered 6×6 matrix M as shown below.

[3]See also Section 5.4.2 in a more general setting.

Vertical bars represent cell boundaries down to the third row.

$$M = \left(\begin{array}{cccccccccc}
0 & \vert & 1 & & 1 & & 1 & 1 & \vert & 2 \\
1 & \vert & 0 & \vert & 1 & \vert & 2 & 2 & \vert & 1 \\
1 & \vert & 1 & \vert & 0 & \vert & 2 & 2 & \vert & ? \\
1 & & 2 & & 2 & & 0 & ? & & ? \\
1 & & 2 & & 2 & & ? & 0 & & ? \\
2 & & 1 & & ? & & ? & ? & & 0
\end{array}\right)$$

•

Suppose that – when using a column or row instantation order – we instantiate the upperdiagonal matrix entry $M_{x,y}$ such that y is the first element of a cell Φ at row x of M. Then we systematically apply the following look-ahead techniques:

- If we assign the value 2 to the matrix entry $M_{x,y}$, then we may temporarily remove the value 1 from the domain of matrix entries $M_{x,z}$ such that $z \in \Phi \setminus \{y\}$. Indeed, assigning 1 to a matrix entry $M_{x,z}$, would violate the lexical ordering constraint imposed on M (see Example 4.10).
- Let $j < r$ such that $M_{j,r} = b \in \{1,2\}$ and $M_{j,y} = c \in \{1,2\}$ with tentative intersection numbers
$$p^{j,r}_{c,1} = k \quad \text{and} \quad p^{j,r}_{c,2} = l.$$
If $|\Phi| = (p^b_{c,1} - k) + (p^b_{c,2} - l)$, then we may temporarily reduce the domain of the matrix entries $M_{r,t}$ to $\{1\}$ and the domain of matrix entries $M_{r,u}$ to $\{2\}$ where $t \in \{y, \ldots, y + p^b_{c,1} - k - 1\}$ and $u \in \Phi \setminus \{y, \ldots, y + p^b_{c,1} - k - 1\}$. Indeed, if we would assign different values to these matrix entries, then either the lexical ordering constraint or the intersection constraint (or both) would be violated (see Example 4.10).
- Before we actually assign a value to the matrix entry $M_{x,y}$, we first provisionally check for each value a of the domain of $M_{x,y}$ in turn, whether the preemptive positive semidefinite check would fail when assigning a to $M_{x,y}$. If so, then the value a can be temporarily removed from the domain of each matrix entry $M_{x,z}$ such that $z \in \Phi$.

Example 4.10. Consider a $\mathrm{srg}(10,3,0,1)$ with corresponding partially filled matrix M as depicted below.

$$
\begin{array}{c|cccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 1 & 0 & 2 & 2 & 1 & 1 & 2 & 2 & 2 & 2 \\
3 & 1 & 2 & 0 & 2 & 2 & 2 & \leftarrow & & & \\
4 & 1 & 2 & 2 & 0 & 2 & 2 & & & & \\
5 & 2 & 1 & 2 & 2 & 0 & 2 & & & & \\
6 & 2 & 1 & \uparrow & & & 0 & & & & \\
7 & 2 & 2 & & & & & 0 & & & \\
8 & 2 & 2 & & & & & & 0 & & \\
9 & 2 & 2 & & & & & & & 0 & \\
10 & 2 & 2 & & & & & & & & 0 \\
\hline
\end{array}
$$

When we extend M by instantiating the matrix entry $M_{3,7}$, then 7 is the first element of the cell $\Phi = \{7, 8, 9, 10\}$ at row 3. The first two look-ahead techniques outlined above can be applied, as follows:

- Clearly if we would assign the value 2 to $M_{3,7}$, then the value 1 can be removed temporarily from the domains of $M_{3,8}$, $M_{3,9}$, $M_{3,10}$.

- Furthermore, note that $M_{1,3} = 1$, $M_{1,7} = 2$, $p^{1,3}_{21} = 0$ while $p^{1}_{21} = 2$ whereas $p^{1,3}_{22} = 2$ while $p^{1}_{22} = 4$. Hence, as $|\Phi| = 4$, we may temporarily reduce the domains of $M_{3,7}$ and $M_{3,8}$ to $\{1\}$ and the domains of $M_{3,9}$ and $M_{3,10}$ to $\{2\}$.

●

Incorporating the look-ahead techniques[4] above into our generation algorithm, our experiments indicate that both the runtime as well as the size of the search tree are substantially smaller when applying a column order canonical form during generation. If we analyze the behaviour of the lexical ordering constraint in detail, we observe an almost complete reduction of its trashing behaviour. Furthermore, we see that the positive semidefiniteness and the rank constraint interact better with the column by column control strategy. The fact that large leading principal submatrices which are fully instantiated turn up faster during search, certainly amounts to this.

[4]Note that similar look-ahead techniques can be developed for the more general case of d-class association schemes

4.4.3 Constraint recording and learning

When we apply Algorithm 4.2 with a $v \times v$ matrix A as input, a witness $x \in \mathbf{R}^{1\times v}$ for A is constructed if A is not positive semidefinite. When using a column by column instantiation order, this witness x allows us to record new constraints. Quite often x is of the form

$$x = (x_1, \ldots, x_k, 0, \ldots, 0, x_v)$$

for some $k \in \{1, \ldots, v-2\}$ such that $x_k = x_v \neq 0$. Let A_k denote the leading principal submatrix of A. Then any $w \times w$ matrix A' with $k < w$ such that

$$(A'_{1,w}, \ldots, A'_{k,w}) = (A_{1,v}, \ldots, A'_{k,v}) \quad \text{and} \quad A_k = A'_k$$

is also not positive semidefinite. Indeed, let $y \in \mathbf{R}^{1\times w}$ such that

$$y = (x_1, \ldots, x_k, 0, \ldots, 0, x_v),$$

then is y a witness for A', as $yA'y^T < 0$.

This leads to the following constraint recording technique which we can incorporate into our generation algorithm. The vector $(A_{1,v}, \ldots, A_{k,v})$ is recorded in a trie Υ [91], until a backtrack occurs to the last instantiated matrix entry of A which contributed to A_k. Note that each entry $A_{l,v}$ with $l < k$ such that $x_l = 0$ is replaced by a wildcard character. Vectors in Υ represent prefixes of upperdiagonal columns of the matrix being generated which can be excluded from consideration. Finally, each time we insert a new vector in Υ, we check whether this vector can be combined with the existing vectors of Υ so as to enable exclusion of shorter prefixes of upperdiagonal columns.

4.5 Generic generator

Throughout this chapter we described our orderly algorithm for the classification of d-class association schemes at a conceptual level. However the practical effectiveness of a generation algorithm is to a large extent determined by the design decisions made when transforming this conceptual description into an explicit generation algorithm. The development of this explicit generation algorithm for d-class association schemes relies on a generic generator which basically applies a recursive backtrack search to generate square symmetric integer matrices with all-zero diagonal of a given order, satisfying some user defined constraints.

A pseudo code version of this generic generator[5] is given in Algorithm 4.3. Before starting the actual generation process, the user should provide a generation description containing:

Matrix The order n of the square symmetric matrices being generated. Internally a $n \times n$ matrix M is constructed with all-zero diagonal and where all non-diagonal matrix entries are still left uninstantiated.

Domain An upperbound d for the domains associated with each uninstantiated matrix entry of M. Internally a $n \times n$ matrix D is constructed in such a way that a non-diagonal matrix entry $D_{x,y}$ consist of an ordered set $\{1, \ldots, d\}$ which represents the domain values of the domain associated with the uninstantiated matrix entry $M_{x,y}$.

Path An object path which determines the order in which matrix of M are instantiated during the during the generation process. The generic generator typically calls the function **prepare** to determine the matrix entry to be instantiated at the next step in the generation process (cf. ❷ in Algorithm 4.3). If this function returns true, then the row and column indices of that entry should be retrieved by invoking the functions **getRow** and **getCol**, respectively. Otherwise, no valid entry could be found. When the generator calls the function **prepare** there may occur internal changes which must systematically be undone by calling the method **restore** (cf. ❸ in Algorithm 4.3). Note that **restore** is not called when **prepare** returned false.

Constraint checkers One or more constraint checkers which are registered in the array checkers. Each constraint checker is responsible for checking whether a certain constraint is satisfied during the generation process. After instantiating a certain matrix entry, the generator forces every registered constraint checker to call the function **setAndCheck** in turn (cf. ❹ in

[5]This generic generator is part of a Java-library which was developed in collaboration with several people of the research group CAAGT. The specific components for the generation of d-class association schemes are ours. See also [106].

Algorithm 4.3). This function determines whether the associated constraint is valid for the current partially instantiated matrix M and if so, to store extra internal information to speed up a future call of **setAndCheck** when further entries are added to the same matrix. In the process of backtracking, the generator is guaranteed to call **unset** for every position for which **setAndCheck** returned true (cf. ❺ in Algorithm 4.3). This enables the checker to rollback all changes in the opposite order in which they have been made by subsequentially invoking **setAndCheck**. Note that **unset** is not called when **setAndCheck** returned false.

Leaf node The user also needs to provide a leaf object which will be activated for each fully instantiated matrix M that passes all constraint checks. Notification is done by calling the method **ship** (cf. ❶ in Algorithm 4.3). An example of a leaf is provided by one which writes fully instantiated matrix M to a file in some user-defined format.

Algorithm 4.3 Generic generator

```
method generate( )
 1: generate(0)

method generate($d$ : int)
 1: if $d$ = leafdepth then                                  ❶
 2:    leaf.ship( )
 3: else if path.prepare($d$) then                           ❷
 4:    $r \leftarrow$ path.getRow( )
 5:    $c \leftarrow$ path.getCol( )
 6:    for all $i \in D_{r,c}$ do
 7:       $M_{r,c} \leftarrow i$
 8:       $M_{c,r} \leftarrow i$
 9:       if checkCheckers($r$, $c$) then
10:          generate($d + 1$)
11:          rollback($r$, $c$, |checkers|)
12:    $M_{r,c} \leftarrow ?$
13:    $M_{c,r} \leftarrow ?$
14:    path.restore($d$)                                     ❸

function checkCheckers($r$, $c$ : int) : boolean
 1: for $i \leftarrow 1 \ldots$ |checkers| do
 2:    if not checkers[$i$].setAndCheck($r$, $c$) then       ❹
 3:       rollback($r$, $c$, $i - 1$)
 4:       return false
 5: return true

method rollback($r$, $c$, $l$ : int)
 1: for $i \leftarrow l \ldots 1$ do
 2:    checkers[$i$].unset($r$, $c$)                         ❺
```

5 Canonicity test

”For many are called, but few are chosen.”[MATTHEW, BIBLE XXII. 14.]

During the orderly generation algorithm we frequently – that is each time a new column becomes completely instantiated – have to check whether a given symmetric $n \times n$ matrix M with zero diagonal and integral offdiagonal entries is in *column order canonical form*. Each such canonicity test computes whether M is the lexicographic minimum of its isomorphism class. Note that this canonicity check is a *decision problem* which merely gives as answer *yes* or *no*, depending on whether the matrix given as input is in column order canonical form or not.

In this Chapter we will discuss the design of an algorithm that does exactly this. We shall develop this algorithm in subsequent stages. At each stage we will introduce the design decisions made and illustrate the impact of each of these decisions with empirical data. This empirical data is mainly obtained from the classification of strongly regular graphs. Most design decisions are however applicable in a broader context than the classification of association schemes. Therefore we present also data obtained from the classification of more general classes of graphs. In Section 5.2 and 5.3 we will first develop the canonicity algorithm only taking into account that the algorithm takes such a matrix M as input, whereas the improvements to the canonicity algorithm introduced in Section 5.4 and 5.5 result from the repeated application of the canonicity algorithm during the course of an orderly generation algorithm.

Remark 5.1. Each column order canonicity test takes a fully instantiated matrix as input. Troughout this chapter we shall denote the set of all integral symmetric $n \times n$ matrix M with zero diagonal by $\mathcal{M}_n$. The notation M_i shall be used as an abbreviation for the $i \times i$ leading principal submatrix of $M \in \mathcal{M}_n$. •

Remark 5.2. Throughout this chapter we shall only consider a column order canonical form. Therefore, to simplify notations, we shall write $A < B$, instead of $A <_c B$ with $A, B \in \mathcal{M}_n$. •

5.1 Representation of permutations groups

In order to describe the canonicity test in this chapter, we need to introduce some additional group-theoretic concepts.

Definition 5.1.1. Consider the right (left) action of a group G onto a set X, then the subgroup

$$G_{x_1,x_2,\ldots,x_i} = (G_{x_1,x_2,\ldots,x_{i-1}})_{x_i}$$

with $\{x_1, x_2, \ldots, x_i\} \subseteq X$ is called the *pointwise stabilizer* of $x_1, x_2, \ldots, x_i$ in G. $\diamond$

Example 5.1. Let $G = \langle (1\,4\,3\,2)\,, (2\,4) \rangle$. Consider the right action of the group G onto the set $X = \{1 \ldots, 4\}$, then the pointwise stabilizer of $1, 3$ in G is $G_{1,3} = \{\text{id}, (2\,4)\}$. •

According to Theorem 3.2.14 there is a one-to-one correspondence between the elements in the orbit of x_i under $G_{x_1,x_2,\ldots,x_{i-1}}$ and the cosets of $G_{x_1,x_2,\ldots,x_i}$ in $G_{x_1,x_2,\ldots,x_{i-1}}$. We have that

$$|x_i^{G_{x_1,x_2,\ldots,x_{i-1}}}| = |G_{x_1,x_2,\ldots,x_{i-1}} : G_{x_1,x_2,\ldots,x_i}|.$$

A fixed set of coset representatives of $G_{x_1,x_2,\ldots,x_i}$ in $G_{x_1,x_2,\ldots,x_{i-1}}$ is denoted by $T^{(i)}$ and is called a *basic transversal*. We have that

$$x_i^{G_{x_1,x_2,\ldots,x_{i-1}}} = \{x_i^t \,:\, t \in T^{(i)}\}.$$

Note that for each $g \in G_{x_1,x_2,\dots,x_{i-1}}$ with $x_i^g = \gamma$ there exists a unique element $h \in G_{x_1,x_2,\dots,x_i}$ such that $g = h\,t$ with $t \in T^{(i)}$ and $x_i^t = \gamma$ with $\gamma \in X \setminus \{x_1, x_2, \dots, x_{i-1}\}$.

Definition 5.1.2. Let G be a group acting right (left) on a set X. Consider a sequence $[x_1, \dots, x_k]$ of elements of X, such that $G_{x_1,\dots,x_k}$ is the trivial group $\{\mathrm{id}\}$. Then

$$G \geq G_{x_1} \geq \dots \geq G_{x_1,\dots,x_k} = \{\mathrm{id}\}$$

is called a *stabilizer chain* for G with *base* $[x_1, \dots, x_k]$. ◇

The pointwise stabilizers in the chain are sometimes denoted by

$$G^{(i)} = G_{x_1,\dots,x_i}$$

and where $G^{(0)} = G$. A subset S of the group G is called a *strong generating set* if S contains generators for each pointwise stabilizer in the chain, i.e. $G^{(i)} = \langle S^{(i)} \rangle$ with

$$S^{(i)} = S \cap G^{(i)}.$$

Note that $S^{(0)} = S$, and hence S generates G. Finally we define

$$\bar{S}^{(i)} = S^{(i)} \setminus S^{(i+1)}$$

with $i \in \{0, \dots, k-1\}$ and $\bar{S}^{(k)} = \{\mathrm{id}\}$.

Example 5.2. Let $\pi_1 = (1\,4\,3\,2)$, $\pi_2 = (2\,4)$ and let the group $G = \langle \pi_1, \pi_2 \rangle$ act right on the set $X = \{1\ \dots, 4\}$. Consider a base $[1, 2]$ for the group G, then a strong generating set S relative to this base is $S = \{\pi_1, \pi_2\}$. The pointwise stabilizers are $G^{(0)} = \langle \pi_1, \pi_2 \rangle$ and $G^{(1)} = \langle \pi_2 \rangle$, while the basic transversals are i.e. $T^{(1)} = \{\mathrm{id}, \pi_1, \pi_1^2, \pi_1^3\}$ and $T^{(2)} = \{\mathrm{id}, \pi_2\}$. •

5.2 The classical algorithm

5.2.1 Traversal of the symmetric group

A first canonicity test (as described in Algorithm 5.1) runs through all permutations $\pi \in \mathrm{Sym}(n)$ and tests whether $\pi M \leq M$. A search through all permutations π of $\mathrm{Sym}(n)$ can be implemented as a backtrack search. Let $i \in \{1, \dots, i\}$.

Define $U^{(i)} = \mathrm{Sym}(n)_{1,\ldots,i}$, i.e., $U^{(i)}$ is the subgroup of $\mathrm{Sym}(n)$ that stabilizes all vertices $1, \ldots, i$ pointwise. We have

$$\mathrm{Sym}(n) = U^{(0)} > U^{(1)} > \cdots > U^{(n-1)} = U^{(n)} = \{\mathrm{id}\} \tag{5.1}$$

with base $[1, \ldots, n]$. A transversal $T^{(i)}$ for $U^{(i)}$ in $U^{(i-1)}$ is given by

$$T^{(i)} = \{\mathrm{id} = (i\,i), (i\,i+1), \ldots, (i\,n)\}. \tag{5.2}$$

Hence each pointwise stabilizer subgroup $U^{(i-1)}$ can be written as a disjoint union of right cosets of the next group $U^{(i)}$ in the chain with only transpositions as coset representatives, that is,

$$U^{(i-1)} = \bigcup_{j=i}^{n} U^{(i)}\ (i\ j). \tag{5.3}$$

It also follows that each $\pi \in U^{(i-1)}$ can be uniquely written as $\pi = u\,(i\,j)$ with $u \in U^{(i)}$ and $i \leq j \leq n$ (take $j = i^{\pi}$). By induction, each permutation $\pi \in U^{(i-1)}$ can be uniquely written as

$$\pi = (n-1\ j_{n-1})\ (n-2\ j_{n-2}) \ldots (k\ j_k) \ldots (i\ j_i) \tag{5.4}$$

with $k \leq j_k \leq n$.

Algorithm 5.1 below provides a straightforward canonicity test based on the above properties. We use the chain of stabilizer subgroups (5.1) to search through all permutations of $\mathrm{Sym}(n)$. A natural way to traverse $\mathrm{Sym}(n) = U^{(0)}$ would be to apply (5.3) to recursively traverse $U^{(i-1)}$. However, it turns out to be advantageous to treat the case $j = i$ separately, and instead successively run through all permutations of $U^{(n-2)} - U^{(n-1)}, \ldots, U^{(0)} - U^{(1)}$, using

$$U^{(i-1)} - U^{(i)} = \bigcup_{j=i+1}^{n} U^{(i)}\ (i\ j). \tag{5.5}$$

Note that $U^{(n-1)} = U^{(n)} = \{id\}$. Algorithm 5.1 is split up into three major parts:

1. The main function **isCanonical(M)** uses the method outlined above to check whether $g\,M < M$ for any $g \in \mathrm{Sym}(n)$. The function returns *false* as soon as a permutation g is found such that $g\,M < M$. Otherwise, the function returns *true*.

2. The function **diffStab(i)** checks whether $g\,M < M$ for any $g \in U^{(i-1)} - U^{(i)}$ by successively traversing all $U^{(i)}(i\,j)$ with $i+1 \leq j \leq n$ (cf. (5.5)). A strictly negative integer is returned as soon as a $g \in U^{(i)}(i\,j)$ is found such that $g\,M < M$. Otherwise if $U^{(i-1)} - U^{(i)}$ does not contain a counterexample to the minimality M, we return a positive integer.

3. The function **rightCoset(π, i)** checks whether $g\,M < M$ for any $g \in U^{(i-1)}\,\pi$. We distinguish between the following cases:
   - If $i \neq n$, then the function successively traverses all $U^{(i)}\,(i\,j)\,\pi$ with $i \leq j \leq n$ (cf. (5.3)). As before, a strictly negative integer is returned as soon as a $g \in U^{(i)}\,(i\,j)\,\pi$ is found such that $g\,M < M$; again if $U^{(i-1)}\,\pi$ does not contain a counterexample to the minimality M, we return a positive integer.
   - Otherwise $U^{(n-1)}\,\pi = \{\pi\}$. This is the base case of the recursive traversal. For each such π, we *effectively test* whether $\pi\,M < M$ (cf. function compare(M, π) in Algorithm 5.1). Moreover, if $\pi \in \mathrm{Aut}\,M$, we store π in a set G_{Aut} (❶ in Algorithm 5.1). Finally, a strictly negative integer is returned if $\pi\,M < M$, otherwise a positive integer is returned.

Note that if M is minimal, the set G_{Aut} contains all automorphisms of M except the identity permutation id.

5.2.2 Partial permutation

Lemma 5.2.1. *Let $g, \pi \in \mathrm{Sym}(n)$. Let $i \in \{1, \ldots, n\}$. Then $\pi \in U^{(i)}g$ if and only if $1^\pi = 1^g$, $2^\pi = 2^g$, $\ldots$, $i^\pi = i^g$.*

Proof : Let $\pi \in U^{(i)}\,g$. Let $x \in \{1, \ldots, i\}$. Write $\pi = u\,g$ with $u \in U^{(i)}$. Then $x^\pi = (x^u)^g = x^g$, because u stabilizes $1, \ldots, i$ pointwise. Conversely, if $x^\pi = x^g$, then $\pi\,g^{-1}$ stabilizes $1, \ldots, i$ pointwise and hence belongs to $U^{(i)}$. ∎

This means that each such coset of $U^{(i)}$ is uniquely characterized by the images of $(1, \ldots, i)$ of any of its elements π. We shall call the i-tuple $(1^\pi, \ldots, i^\pi)$ the

Algorithm 5.1 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function isCanonical($M \in \mathcal{M}_n$) : boolean
for $i \leftarrow n-1, \ldots, 1$ do
  if diffStab($i$) $< 0$ then
    return false
return true

function diffStab($i$ : int) : int
for $j \leftarrow i+1, \ldots, n$ do
  if rightCoset($(i\ j)$, $i+1$) $< 0$ then
    return $-1$
return 1

function rightCoset($\pi \in \mathrm{Sym}(n)$, $i$ : int) : int
if $i = n$ then
  $d \leftarrow$ compare($\pi$)
  if $d = 0$ then
    $G_{\mathrm{Aut}} \leftarrow G_{\mathrm{Aut}} \cup \pi$                    ❶
  return $d$
else
  for $j \leftarrow i, \ldots, n$ do
    $\pi' \leftarrow (i\ j)\ \pi$
    if rightCoset($\pi'$, $i+1$) $< 0$ then
      return $-1$
  return 1

function compare ($\pi \in \mathrm{Sym}(n)$) : int
for $i \leftarrow 2, \ldots, n$ do
  for $j \leftarrow 1, \ldots, i-1$ do
    if $M_{i^\pi, j^\pi} < M_{i,j}$ then
      return $-1$
    else if $M_{i^\pi, j^\pi} > M_{i,j}$ then
      return 1
return 0
```

prefix of π of size i. In other words : two permutations belong to the same right coset of $U^{(i)}$ if and only if they have the same prefix of size i.

Lemma 5.2.2. *Let $g \in \mathrm{Sym}(n)$, $u \in U^{(i)}$. Let $i \in \{1, \ldots, n\}$. Then $(u\,g\,M)_i = (g\,M)_i$. In particular, $(u\,M)_i = M_i$.*

Proof : Let x, $y \in \{1, \ldots, i\}$. Then $(M_i)_{x,y} = M_{x,y}$ and $((u\,g\,M)_i)_{x,y} = (u\,g\,M)_{x,y} = (M)_{x^{u\,g}, y^{u\,g}} = (M)_{x^g, y^g} = (g\,M)_{x,y} = ((g\,M)_i)_{x,y}$, because u stabilizes $1, \ldots, i$ pointwise. ∎

It follows that $(\pi\,M)_i$ is determined by the right coset of $U^{(i)}$ to which π belongs, i.e., by the prefix of π of length i. A more informal way to see this is the following: in order to determine the matrix entry $(\pi\,M)_{x,y}$ when $x, y \leq i$, we only need to know the values of $1^\pi, 2^\pi, \ldots, i^\pi$.

Corollary 5.2.1. *Let $\pi \in \mathrm{Sym}(n)$. Let $i \in \{1, \ldots, n\}$ such that $(\pi\,M)_i > M_i$. Then $g\,M > M$ for each $g \in U^{(i)}\,\pi$.* ◇

Whenever $M \in \mathcal{M}_n$ is in canonical form, Algorithm 5.1 runs through all $n!$ permutations of $\mathrm{Sym}(n)$. Algorithm 5.2 provides an improvement based on the above corollary. The differences between Algorithm 5.2 and 5.1 are essentially the following:

1. The function **rightCoset(**π**,** i**)** is only called if $(\pi\,M)_{i-1} = M_{i-1}$ (❶ in Algorithm 5.2). Indeed, using Corollary 5.2.1, if $(\pi\,M)_{i-1} > M_{i-1}$, then $g\,M > M$ for each $g \in U^{(i-1)}\,\pi$. Hence no such g would lead to a counterexample of the minimality of M and we can consequently discard the entire right coset $U^{(i-1)}\,\pi$. If $(\pi\,M)_{i-1} < M_{i-1}$, then π is a counterexample of the minimality of M. As before, we then can terminate the canonicity test (❷ in Algorithm 5.2).

2. The function **compare(**π**,** i**)** takes an additional parameter i. It now checks whether $(\pi\,M)_i$ is smaller, equal or greater than M_i (❸ in Algorithm 5.2). We may assume that the function **compare** is only called when $(\pi\,M)_{i-1} = M_{i-1}$. Otherwise, let k be the smallest integer such that $(\pi\,M)_k \neq M_k$. If $(\pi\,M)_k < M_k$, then the canonicity test would have been terminated earlier as we would have found a counterexample to the minimality of M whereas if $(\pi\,M)_k > M_k$ the entire right coset $U^{(k)}\,\pi \supset U^{(i)}\,\pi$ would have been discarded. As a result of this, it suffices to compare $(\pi\,M)_{1\ldots i-1,i}$ with $M_{1\ldots i-1,i}$, instead of $(\pi\,M)_i$ with M_i.

If M is minimal, then as in Algorithm 5.1, the set G_{Aut} contains all automorphisms of M. After all, $(g\,M)_i = M_i$ for each $g \in U^{(i)}\,h$ with $h \in \mathrm{Aut}\,M$.

Algorithm 5.2 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

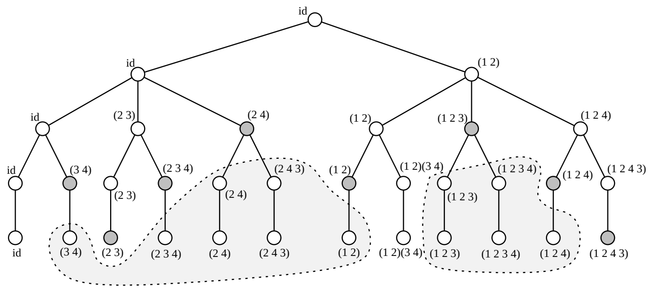
```
**function** isCanonical($M \in \mathcal{M}_n$) : boolean
**for** ($i \leftarrow n-1,\ldots 1$) **do**
    **if** diffStab($i$) $< 0$ **then**
        **return** false
**return** true

**function** diffStab($i$ : int) : int
**for** $j \leftarrow i+1,\ldots,n$ **do**
    $d \leftarrow$ compare($(i\ j)$, $i$)                         ❸
    **if** $d < 0$ **then**                                        ❷
        **return** $d$
    **else if** $d = 0$ **then**                                   ❶
        $d \leftarrow$ rightCoset($(i\ j)$, $i+1$)
        **if** $d < 0$ **then**
            **return** $d$
**return** 1

**function** rightCoset($\pi \in \mathrm{Sym}(n)$, $i$ : int) : int
**if** $i = n+1$ **then**
    $G_{\mathrm{Aut}} \leftarrow G_{\mathrm{Aut}} \cup \pi$
    **return** 0
**else**
    **for** $j \leftarrow i,\ldots,n$ **do**
        $\pi' \leftarrow (i\ j)\ \pi$
        $d \leftarrow$ compare($\pi'$, $i$)                     ❸
        **if** $d < 0$ **then**                                    ❷
            **return** $d$
        **else if** $d = 0$ **then**                               ❶
            $d \leftarrow$ rightCoset($\pi'$, $i+1$)
            **if** $d < 0$ **then**
                **return** $d$
    **return** 1
```

Algorithm 5.2 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function compare ($\pi \in \mathrm{Sym}(n)$, $i$ : int) : int
1: for $j \leftarrow 1, \dots, i-1$ do
2:     if $M_{i^\pi, j^\pi} > M_{i,j}$ then
3:         return $j$
4:     else if $M_{i^\pi, j^\pi} < M_{i,j}$ then
5:         return $-j$
6: return 0
```

Example 5.3. Consider the matrix $M \in \mathcal{M}_4$ below. The recursion tree shown, corresponds to the traversal of both $U^{(1)}$ and $U^{(1)}\,(1\ 2)$ by Algorithm 5.1. For each node at depth d in the recursion, the coset representative of the corresponding right coset $U^{(d)}\,\pi$ is denoted. The right cosets situated in one of the light gray areas correspond to the right cosets which are additionally discarded by Algorithm 5.2. More precisely, as soon as Algorithm 5.2 considers a right $U^{(d)}\pi$ such that $(\pi\, M)_d > M_d$, the recursion tree is pruned (the corresponding nodes are colored light gray).

$$M = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}$$



●

5.2.3 Analysis and empirical data

We will consider the *worst-case* time complexity of Algorithm 5.1 and Algorithm 5.2. If $M \in \mathcal{M}_n$ is minimal, Algorithm 5.1 traverses all $n!$ permutations of $\mathrm{Sym}(n)$. Otherwise, it terminates prematurely, that is, as soon as a permutation π such that $\pi\, M < M$ is encountered. Hence the worst-case clearly occurs when M is in canonical form. The corresponding recursion tree then contains exactly $[U^{(0)} : U^{(d)}] = n!/(n-d)!$ nodes at depth d, i.e., the number of disjoint right cosets of $U^{(d)}$ in $\mathrm{Sym}(n)$. For each π we encounter, we have to check at its corresponding leaf node whether $\pi M < M$. This costs at most $\frac{n(n-1)}{2}$ matrix entry comparisons, that is, if $\pi M = M$. Hence the worst-case time complexity of Algorithm 5.1 is $O(n!\, n^2)$. Algorithm 5.2 will generally traverse only a fraction of $\mathrm{Sym}(n)$. For each right coset $U^{(d)}\pi$ considered at depth d in the recursion, the algorithm checks whether $(\pi M)_{1\ldots d-1,d} = M_{1\ldots d-1,d}$. This costs at most $d-1$ matrix entry comparisons. If $(\pi M)_{1\ldots d-1,d} \neq M_{1\ldots d-1,d}$ then either $U^{(d)}\pi$ can be discarded entirely or the canonicity test can even be terminated prematurely. The worst-case clearly occurs when M is minimal and the recursion tree is never pruned. The total cost is then given by

$$\sum_{d=1}^{n} \frac{n!\,(d-1)}{(n-d)!} \le (n-1)\,n! \sum_{k=0}^{n-1} \frac{1}{k!} \le (n-1)\,n!\,e \le n!\,n\,e$$

as $\sum_{k=0}^{\infty} \frac{1}{k!} = e$ [92]. Hence the worst-case time complexity of Algorithm 5.2 is $O(n!\,n)$. It is no surprise that the worst-case time complexity of Algorithm 5.2 is better than that of Algorithm 5.1. Indeed, even when the recursion tree is never pruned, Algorithm 5.2 takes advantage of the fact that the certificate $\mathcal{C}(\pi M)$ and $\mathcal{C}(\pi' M)$ have the same prefix of length $\frac{l(l-1)}{2}$ whenever the permutations π and π' have the same prefix of length $l \in \{1 \ldots n\}$. Although the worst-case for Algorithm 5.2 only occurs when M is minimal and the recursion tree is never pruned, the analysis suggests that if M has a large automorphism, the size of the recursion tree will tend to increase. However also the structure of M will determine the number of times and the depth at which we can prune the recursion tree.

The effectiveness of the in Algorithm 5.2 introduced ”partial permutation”criterion is examined by comparing data obtained from the orderly generation of simple graphs of a given order v, regular graphs of a given order v and degree k and strongly regular graphs with certain (v, k, λ, μ) parameter sets. In Table 5.1 the total number of graphs generated by the orderly algorithm, along with total number of permutations (partial or complete) checked by all canonicity tests which have been executed, is denoted. The empirical data obtained illustrates that the application of Algorithm 5.1 during an orderly generation algorithm turns out to be infeasible for graphs of relatively small small order. Just merely the application of the canonicity test for each graph being generated, requires the traversal of $v!$ permutations.

The in Algorithm 5.2 introduced pruning criterion turns out to be an enormous improvement over Algorithm 5.1. The total number of permutations considered, is drastically smaller and increases much more slowly with increasing order of v. The empirical data shows that mostly only a fraction of the full recursion tree is effectively considered. This makes the application of the canonicity test of Algorithm 5.2 in the orderly generation of graphs feasible for graphs of significantly larger order.

					Algorithm 5.1	Algorithm 5.2
v	k	λ	μ	$\exists$	No. of perm. checked	No. of perm. checked
8	–	–	–	12346	1 460 991 941	11 686 005
9	–	–	–	274668	-	414 180 087
9	4	–	–	16	24 418 500	93 267
9	6	–	–	4	4 677 199	52154
10	4	–	–	60	982 122 577	740 636
11	4	–	–	266	5 364 303 086	5 123 710
11	6	–	–	266	-	11 825 305
5	2	0	1	1	393	120
9	4	1	2	1	1 112 028	2 918
10	3	0	1	1	10 976 118	5 038
10	6	3	4		10 976 178	8 658
13	6	2	3	1	1 834 834 209	19 331
15	6	1	3	1	-	80 932
15	8	4	4		-	130 624

Tabel 5.1: Comparison of Algorithm 5.1 and 5.2 applied in the orderly generation of simple, regular and strongly regular graphs.

The effect of the *size of the automorphism group* of a graph (which is in column order canonical form) on the total number of permutations checked by Algorithm 5.2 is examined. A selection of the data obtained is depicted in Figure 5.1. Based on this empirical data we can observe, in the case of simple and regular graphs, an apparent correlation between the size of the automorphism group and the number of nodes in the corresponding recursion tree. The number of nodes generally tends to increase as the size of the automorphism group increases. However, for a given automorphism group size, the number of nodes in the recursion tree can differ significantly. Based on the canonicity tests on minimal representatives of strongly regular graphs, we finally can observe that the impact on the size of the recursion tree is only apparent for those graphs graphs with the largest automorphism group. After all, not only the size of the automorphism group determines the size of the recursion, the graph itself plays an important role in the number of times and the depth at which the recursion tree is pruned.

5.3 Using the automorphism group

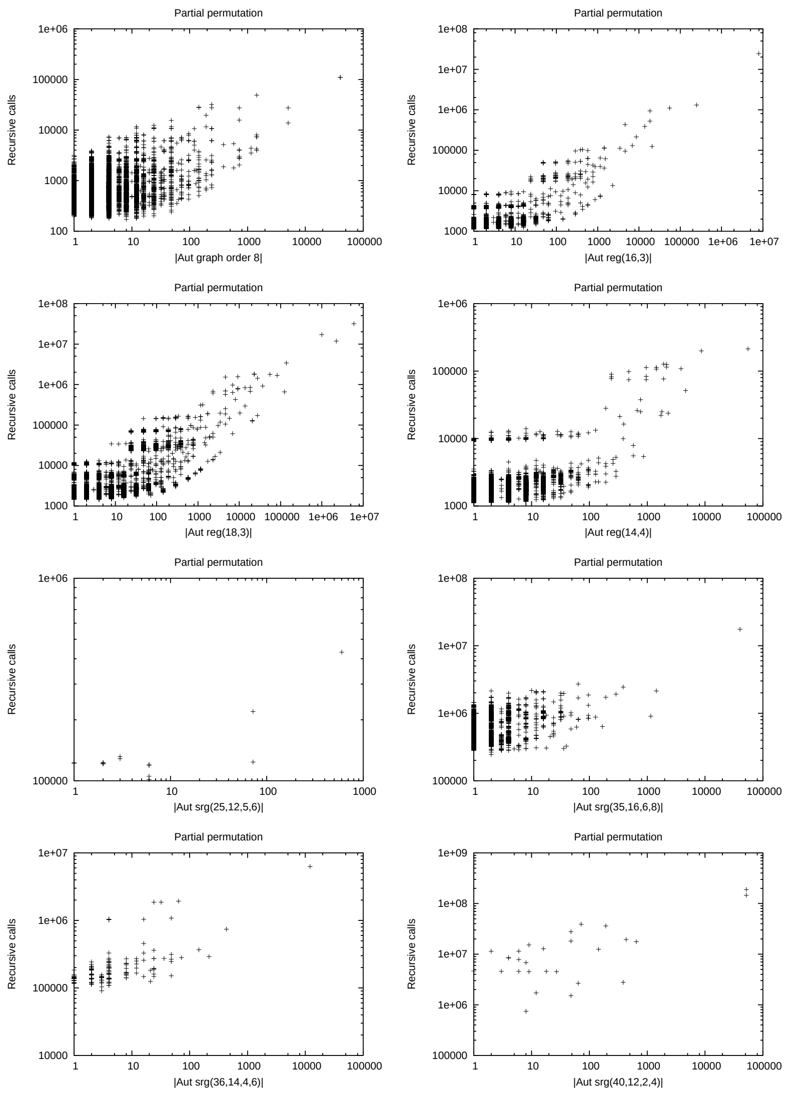
In this section we shall introduce additional pruning criteria which ensure us that not the full automorphism group of M has to be traversed. Automorphisms of M encountered so far during traversal are employed in two distinct manners in pruning the recursion tree.

5.3.1 Discovering a new automorphism

Lemma 5.3.1. *Let $i \in \{1 \ldots n\}$. Let $g \in \operatorname{Aut} M$. If $u\,M \geq M$ for each $u \in U^{(i)}$, then also $h\,M \geq M$ for each $h \in U^{(i)}\,g$.*

Proof : Each $h \in U^{(i)}\,g$ can uniquely be written as $h = u\,g$ for some $u \in U^{(i)}$. Hence $h\,M = u\,(g\,M) = u\,M \geq M$ as $g\,M = M$. ■

This lemma provides a first improvement to Algorithm 5.2 as outlined in Algorithm 5.3. The function **rightCoset(**π, i, s**)** takes an additional parameter s. This parameter refers to the stabilizer group difference $U^{(s)} - U^{(s+1)}$ of which the right coset $U^{(i-1)}\,\pi$ is a subset. When the function is called with $i = n + 1$, an automorphism π is discovered. We then can discard the entire remaining part of the right coset $U^{(s+1)}\,\pi$. The extra equality (❷ in Algorithm 5.3) ensures us that the remainder of $U^{(s+1)}\,\pi$ is pruned. Clearly $u\,M \geq M$ for each $u \in U^{(s+1)}$, otherwise the canonicity test would have been terminated preceding the traversal of $U^{(s)} - U^{(s+1)}$. Using Lemma 5.3.1 we find that $h\,M \geq M$



Figuur 5.1: Number of recursive calls of Algorithm 5.2 for minimal representatives of simple, regular and strongly regular graphs.

Algorithm 5.3 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function isCanonical($M \in \mathcal{M}_n$) : boolean
 1: for $i \leftarrow n-1, \dots, 1$ do
 2:    if diffStab($i$) $< 0$ then
 3:       return false
 4: return true

function diffStab($i$ : int) : int
 1: for $j \leftarrow i+1, \dots, n$ do
 2:    $d \leftarrow$ compare($(i\,j)$, $i$)
 3:    if $d < 0$ then
 4:       return $d$
 5:    else if $d = 0$ then
 6:       $d \leftarrow$ rightCoset($(i\,j)$, $i+1$, $i-1$)
 7:       if $d < 0$ then
 8:          return $d$
 9: return 1

function rightCoset($\pi \in U^{(s)} - U^{(s+1)}$, $i$, $s$ : int) : int
 1: if $i = n+1$ then
 2:    $T^{(s+1)}_{\mathrm{Aut}} \leftarrow T^{(s+1)}_{\mathrm{Aut}} \cup \pi$                    ❶
 3:    return 0
 4: else
 5:    for $j \leftarrow i, \dots n$ do
 6:       $\pi' \leftarrow (i\ j)\ \pi$
 7:       $d \leftarrow$ compare($\pi'$, $i$)
 8:       if $d < 0$ then
 9:          return $d$
10:       else if $d = 0$ then
11:          $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
12:          if $d < 0 \vee \mathbf{d} = \mathbf{0}$ then                 ❷
13:             return $d$
14:    return 1

function compare ($\pi \in \mathrm{Sym}(n)$, $i$ : int) : int
 1: {cf. function "compare" in Algorithm 5.2}
```

for each $h \in U^{(s+1)}\,\pi$. Hence no permutation of $U^{(s+1)}\,\pi$ could lead to a counterexample to the minimality of M.

Moreover, π is now stored in the set $T_{\text{Aut}}^{(s+1)}$ (❶ in Algorithm 5.3). This set is initially empty, but will contain all automorphisms of M encountered during the traversal of the difference $U^{(s)} - U^{(s+1)}$. If M is minimal, both Algorithm 5.1 and Algorithm 5.2 have as a side effect that all automorphisms of M are obtained during traversal. This is no longer true in Algorithm 5.3.

Let $i \in \{1, \ldots, n\}$. Define the group

$$(\operatorname{Aut} M)^{(i)} = (\operatorname{Aut} M)_{1,\ldots,i} \tag{5.6}$$

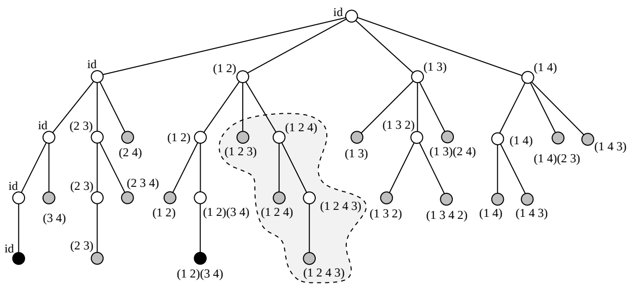
i.e., $(\operatorname{Aut} M)^{(i)}$ is the subgroup of $\operatorname{Aut} M$ that stabilizes, like the group $U^{(i)}$, all vertices $1, \ldots, i$ pointwise. As a consequence of the following proposition, we find that if M is minimal, Algorithm 5.3 obtains for each $s \in \{0, \ldots, n-2\}$ a transversal of $(\operatorname{Aut} M)^{(s+1)}$ in $(\operatorname{Aut} M)^{(s)}$.

Proposition 5.3.1. *Let $i \in \{0, \ldots, n-2\}$. If $u\,M \geq M$ for each $u \in U^{(i)}$, then the set $T_{\text{Aut}}^{(i+1)} \cup \{\text{id}\}$ obtained by Algorithm 5.3 is a transversal of $(\operatorname{Aut} M)^{(i+1)}$ in $(\operatorname{Aut} M)^{(i)}$.*

Proof : Algorithm 5.3 traverses all right cosets $U^{(i+1)}\,(i+1\;j)$ with $j \in \{i+2, \ldots, n\}$, after all $u\,M \geq M$ for each $u \in U^{(i)}$. Consider such a right coset $U^{(i+1)}\,(i+1\;k)$ and assume its intersection with $\operatorname{Aut} M$ is non-empty. Then exactly one element π of this intersection is added to the set $T_{\text{Aut}}^{(i+1)}$ and the remaining permutations of $U^{(i+1)}\,(i+1\;k)$ are discarded. If $h \in \operatorname{Aut} M$ is discarded at this point, then $h\,\pi^{-1} \in \operatorname{Aut} M$ and $h\,\pi^{-1} \in U^{(i)}$. Hence also $h \in (\operatorname{Aut} M)^{(i+1)}\,\pi$. Therefore the set $T_{\text{Aut}}^{(i+1)} \cup \{\text{id}\}$ contains one representative for every right coset of $(\operatorname{Aut} M)^{(i+1)}$ in $(\operatorname{Aut} M)^{(i)}$. ∎

Example 5.4. Consider again the matrix $M \in \mathcal{M}_4$ below. The recursion tree shown, corresponds to the traversal of $\operatorname{Sym}(4)$ by Algorithm 5.2.

$$M = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}$$



When Algorithm 5.3 traverses all elements of $U^{(1)}$ it encounters no counter-examples to the minimality of M. Then upon the discovery of the automorphism $(1\,2)\,(3\,4) \in U^{(1)}\,(1\,2)$ (its corresponding node in the recursion tree is colored black), the entire remaining part of $U^{(1)}\,(1\,2)$ can be discarded (cf. light grey area). $\bullet$

5.3.2 Minimal in orbit

A second improvement makes use of the automorphisms of M encountered previously during traversal.

Lemma 5.3.2. *Let $i \in \{1,\dots,n\}$ and $\varphi \in (\operatorname{Aut} M)^{(i-1)}$. Let π, $\sigma \in U^{(i-1)}$ such that $g\,M \geq M$ for each $g \in U^{(i)}\,\pi$. If $i^{\sigma} = i^{\pi\,\varphi}$, then $h\,M \geq M$ for each $h \in U^{(i)}\,\sigma$.*

Proof : Because π, σ and $\varphi \in U^{(i-1)}$, we find that $j^{\pi\,\varphi} = j^{\sigma}$ for each $j \in \{1,\dots,i-1\}$. Moreover, also $i^{\sigma} = i^{\pi\,\varphi}$. Hence $\pi\,\varphi$ and σ have the same i-prefix. Therefore $U^{(i)}\,\sigma = U^{(i)}\,\pi\,\varphi$ and each $h \in U^{(i)}\,\sigma$ can uniquely be written as $h = u\,\pi\,\varphi$ for some $u \in U^{(i)}$. Hence $h\,M = u\,\pi\,\varphi\,M = u\,\pi\,M \geq M$ as $\varphi \in \operatorname{Aut} M$. ■

Corollary 5.3.1. *Let $i \in \{1,\dots,n\}$, $j \in \{i,\dots,n\}$ and π, $\sigma \in U^{(i-1)}$ such that $i^{\pi} = j$. Let $H \leq (\operatorname{Aut} M)^{(i-1)}$ and $g\,M \geq M$ for each $g \in U^{(i)}\,\pi$. If $i^{\sigma} \in j^{H}$, then $h\,M \geq M$ for each $h \in U^{(i)}\,\sigma$.* $\diamond$

This corollary provides an improvement to Algorithm 5.3, as outlined in Algorithm 5.4. The main differences are in essence the following:

1. When the function **rightCoset(**π, i, s**)** is called with $i = n+1$, an automorphism $\pi \in (\operatorname{Aut} M)^{(s)}$ is discovered which is now stored in the set $\bar{S}^{(s)}_{\operatorname{Aut} M}$ (❶ in Algorithm 5.4). For notational simplicity we set the set

$$S^{(s)}_{\operatorname{Aut} M} = \bigcup_{q=n-2}^{s} \bar{S}^{(q)}_{\operatorname{Aut} M}. \tag{5.7}$$

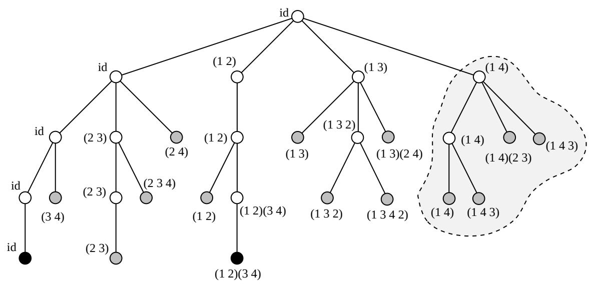
2. The function **diffStab**(i) runs through $U^{(i-1)}$-$U^{(i)}$ by successively traversing the right cosets

$$U^{(i)}\,(i\;i+1),\, U^{(i)}\,(i\;i+2), \ldots, U^{(i)}\,(i\;n).$$

   Preceding the traversal of such a right coset $U^{(i)}\,(i\;j)$ with $j \in \{i+1, \ldots, n\}$, we now additionally check whether j is the minimal element in its orbit under $H = \langle S^{(i-1)}_{\operatorname{Aut} M} \rangle \leq (\operatorname{Aut} M)^{(i-1)}$, i.e., the group generated by the so far discovered automorphisms of M (❷ in Algorithm 5.4). If $k = \min(j^H) < j$, then we discard the entire right coset $U^{(i)}\,(i\;j)$ from traversal. After all, by the structure of the algorithm, $h\,M \geq M$ for each $h \in U^{(i)}\,(i\;l)$ with $l \in \{i, \ldots, j-1\}$, among which $U^{(i)}\,(i\;k)$. Hence according to Corollary 5.3.1 we find that $g\,M \geq M$ for each $g \in U^{(i)}\,(i\;j)$. Hence no such permutation g could ever lead to a counterexample of the minimality of M.

Example 5.5. Consider the matrix $M \in \mathcal{M}_4$ below.

$$M = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}$$



When Algorithm 5.4 has traversed all permutations of $U^{(1)}$, $U^{(1)}\,(1\,2)$ and $U^{(1)}\,(1\,3)$, no counterexample to the minimality of M has been found. At that point $S^{(0)}_{\operatorname{Aut} M} = \{(1\,2)\,(3\,4)\}$. Let $H = \langle S^{(0)}_{\operatorname{Aut} M} \rangle$ and $X = \{1, \ldots, 4\}$, then $H \mathbin{\backslash\backslash} X =$

Algorithm 5.4 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function isCanonical($M \in \mathcal{M}_n$) : boolean
 1: for $i \leftarrow n-1, \dots 1$ do
 2:    if diffStab($i$) $< 0$ then
 3:       return false
 4: return true

function diffStab($i$ : int) : int
 1: for $j \leftarrow i+1, \dots n$ do
 2:    if $j = \min(j^{\langle S^{(i-1)}_{\mathrm{Aut}M} \rangle})$ then                ❷
 3:       $d \leftarrow$ compare($(i\ j)$, $i$)
 4:       if $d < 0$ then
 5:          return $d$
 6:       else if $d = 0$ then
 7:          $d \leftarrow$ rightCoset($(i\ j)$, $i+1$, $i-1$)
 8:          if $d < 0$ then
 9:             return $d$
10: return 1

function rightCoset($\pi \in U^{(s)} - U^{(s+1)}$, $i, s$ : int) : int
 1: if $i = n+1$ then
 2:    $\bar{S}^{(s)}_{\mathrm{Aut}\,M} \leftarrow \bar{S}^{(s)}_{\mathrm{Aut}\,M} \cup \pi$                ❶
 3:    return 0
 4: else
 5:    for $j \leftarrow i, \dots, n$ do
 6:       $\pi' \leftarrow (i\ j)\ \pi$
 7:       $d \leftarrow$ compare($\pi'$, $i$)
 8:       if $d < 0$ then
 9:          return $-d$
10:       else if $d = 0$ then
11:          $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
12:          if $d < 0 \vee d = 0$ then
13:             return $d$
14:    return 1

function compare ($\pi \in \mathrm{Sym}(n)$, $i$ : int) : int
 1: {cf. function "compare" in Algorithm 5.2}
```

$\{\{1, 2\}, \{3, 4\}\}$. According to Corollary 5.3.1 we can now discard the entire right coset $U^{(1)}\,(1\,4)$, as 4 is not the minimal element in its orbit under H (cf. grey area). •

Algorithm 5.3 has as a side effect that when M is minimal, a transversal of $(\operatorname{Aut} M)^{(s+1)}$ in $(\operatorname{Aut} M)^{(s)}$ is obtained for each $s \in \{n-2, \ldots, 0\}$ (Proposition 5.3.1). This is no longer true for Algorithm 5.4. As a consequence of the following proposition we find that when M is minimal, Algorithm 5.4 obtains a strong generating set $S^{(0)}_{\operatorname{Aut} M}$ for the automorphism group $\operatorname{Aut} M$ with base $[1, \ldots, n]$ like in (5.1).

Proposition 5.3.2. *Let $i \in \{2, \ldots, n\}$. If $u\,M \geq M$ for each $u \in U^{(n-i)}$, then the set $S^{(n-i)}_{\operatorname{Aut} M}$ obtained by Algorithm 5.4 is a strong generating set for $(\operatorname{Aut} M)^{(n-i)}$.*

Proof : We shall prove this proposition by induction on i. Without loss of generality we may assume that Algorithm 5.4 traverses the stabilizer group $U^{(n-1)}$ prior to the stabilizer difference $U^{(n-2)} - U^{(n-1)}$. Let $i = 1$. Since both $U^{(n-i)}$ and $(\operatorname{Aut} M)^{(n)}$ are trivial and n is the minimal element in $n^{(\operatorname{Aut} M)^{(n)}}$, we find that $(\operatorname{Aut} M)^{(n-1)} = \langle S^{(n-1)}_{\operatorname{Aut} M} \rangle = \langle \mathrm{id} \rangle$.

Assume this proposition is valid for $i = p \in \{2, \ldots, n-1\}$. Let $k = n - p$. Hence $(\operatorname{Aut} M)^{(k)} = \langle S^{(k)}_{\operatorname{Aut} M} \rangle$. For $i = p + 1$ Algorithm 5.4 first traverses the stabilizer group $U^{(k)}$ and then successively the right cosets

$$U^{(k)}\,(k\,k+1), U^{(k)}\,(k\,k+2), \ldots, U^{(k)}\,(k\,n). \tag{5.8}$$

Let $H = \langle S^{(k-1)}_{\operatorname{Aut} M} \rangle$. Consider a right coset $U^{(k)}\,(k\;j)$ with $j \in \{k+1, \ldots, n\}$. Distinguish between the following cases:

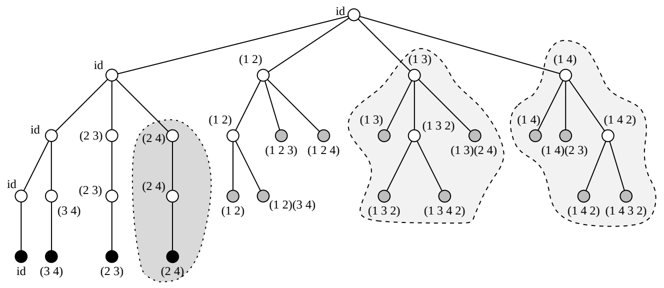
1. If $j = \min(j^H)$ then $U^{(k)}\,(k\,j)$ is traversed. Assume $U^{(k)}\,(k\,j)$ contains one or more automorphisms of M. Let π be the first automorphism discovered in $U^{(k)}\,(k\;j)$ and let $H' = \langle S^{(k-1)}_{\operatorname{Aut} M}, \pi \rangle$. Then π is added to $\bar{S}^{(k-1)}_{\operatorname{Aut} M}$, the remaining permutations of $U^{(k)}\,(k\;j)$ are neglected and the traversal continues at the next coset. For each $h \in U^{(k)}\,(k\;j)$ such that also $h \in \operatorname{Aut} M$ we have $h \in (\operatorname{Aut} M)^{(k)}\,\pi$ according to Proposition 5.3.2. Hence $h \in H'$.
2. Otherwise, let $l = \min(j^H)$, then we discard $U^{(k)}\,(k\;j)$ from traversal. Suppose $U^{(k)}\,(k\,j)$ contains a $\pi \in \operatorname{Aut} M$. Since $l \in j^H$ we have $l^h = j$

where $h \in H$. As $(k\ l)\,h \in U^{(k-1)}\,(k\ j)$ we have $\pi\,M = u\,(k\ l)\,h\ M = u\,(k\ l)\,M = M$ for some $u \in U^{(k-1)}$. As $l = \min(j^H)$ the right coset $U^{(k)}\,(k\ l)$ was considered prior to the right coset $U^{(k)}\,(k\ j)$. We therefore find that $u\,(k\ l) \in H$ (cf. first case). Hence also $\pi \in H$.

After the traversal of (5.8), we have for each automorphism $h \in U^{(k)}\,(k\,m)$ with $m \in \{k+1,\ldots,n\}$ that $h \in H = \langle S^{(k-1)}_{\operatorname{Aut} M}\rangle$. Since $(\operatorname{Aut} M)^{(k)} = \langle S^{(k)}_{\operatorname{Aut} M}\rangle$ we therefore find that $(\operatorname{Aut} M)^{(k-1)} = \langle S^{(k-1)}_{\operatorname{Aut} M}\rangle$. Hence this proposition is also valid for $i = p+1$. ■

Example 5.6. Consider the matrix $M \in \mathcal{M}_4$ below. When Algorithm 5.4 has traversed all permutations of $U^{(2)}$ and $U^{(1)}\,(2\,3)$, no counterexamples to the minimality of M have been found. Moreover, $\bar{S}^{(2)}_{\operatorname{Aut} M} = \{(3\ 4)\}$ whereas $\bar{S}^{(3)}_{\operatorname{Aut} M} = \{(2\ 3)\}$. Let $H = \langle S^{(2)}_{\operatorname{Aut} M}\rangle$ and $X = \{1,\ldots,4\}$, then $H \,\backslash\!\backslash\, X = \{\{1\}, \{2, 3, 4\}\}$. According to Corollary 5.3.1 we can now discard the entire right coset $U^{(2)}\,(2\,4)$, as 4 is not the minimal element in its orbit under H (cf. dark grey area). Note that the transposition $(2\,4)$ is an automorphism of M. We leave it to the reader to verify that $(2\,4) \in H$.

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 2 & 0 & 2 \\ 1 & 2 & 2 & 0 \end{pmatrix}$$



When Algorithm 5.4 has traversed all permutations of $U^{(1)}$ and $U^{(1)}\,(1\,2)$, no counterexamples to the minimality of M have been found. According to Corollary 5.3.1 we can discard the entire right cosets $U^{(1)}\,(1\,3)$ and $U^{(1)}\,(1\,4)$, as both 3 and 4 are not the minimal elements in their orbit under H (cf. light grey area). •

5.3.3 Maintaining an orbit partition

The incorporation of the “minimal in orbit” criterion into our canonicity algorithm, requires the facility to check whether a given element is minimal in its orbit under a group H generated by the so far encountered automorphisms of $M \in \mathcal{M}_n$. In order to do so we keep track of the corresponding orbit partition P. This partition should be updated each time a new automorphism is encountered. The orbit partition P is represented as a *forest* $F(P)$ with vertex set $X = \{1, \ldots, n\}$ such that each cell of P corresponds to a unique *tree* in $F(P)$. At the same time an array $M(P)$ of integers is maintained. The value $M(P)[x]$ at index x in this array is equal to $\min(x^H)$. The main benefit of this array is that it allows to check in constant time whether a given element is minimal in its orbit. After all, an element x in minimal in its orbit if and only if $M(P)[x] = x$.

Initially P is discrete. Each time Algorithm 5.4 encounters an automorphism $\pi \in U^{(k)} - U^{(k+1)}$ with $k \in \{0, \ldots, n-2\}$, we add π to $\bar{S}^{(k)}_{\mathrm{Aut}\,M}$. Before we actually add π to $\bar{S}^{(k)}_{\mathrm{Aut}\,M}$, we now first have to update the orbit partition

$$P = \langle S^{(k)}_{\mathrm{Aut}\,M} \rangle \mathbin{\backslash\!\backslash} X, \tag{5.9}$$

i.e., refine P to the orbit partition

$$Q = \langle S^{(k)}_{\mathrm{Aut}\,M} \cup \{\pi\} \rangle \mathbin{\backslash\!\backslash} X. \tag{5.10}$$

This refinement essentially amounts to determining the set of orbits in P which merge under the action of π. To put it differently, the resulting orbit partition Q can be seen as the union of the orbit partitions P and $\langle \{\pi\} \rangle \mathbin{\backslash\!\backslash} X$. The orbit partition $\langle \{\pi\} \rangle \mathbin{\backslash\!\backslash} X$ can quite easily be obtained by decomposing π into disjoint cycles, which demands time linear in $|X|$. More precisely, decomposing π requires $n - k$ image computations. Let $l \leq |\langle \{\pi\} \rangle \mathbin{\backslash\!\backslash} X| - k$, then π can be written as a product of disjoint cycles

$$\pi = \left(x_{1,1}\ x_{2,1} \ldots x_{p_1,1}\right)\ \left(x_{1,1}\ x_{2,2} \ldots x_{p_2,2}\right)\ \ldots\ \left(x_{1,l}\ x_{2,l} \ldots x_{p_l,l}\right) \tag{5.11}$$

where $\bigcup_{i=1}^{l} \bigcup_{j=1}^{p_i} x_{j,i} \subseteq \{k+1, \ldots, n\}$ and the elements which make up a cycle correspond to a unique cell of $\langle \{\pi\} \rangle \mathbin{\backslash\!\backslash} X$. We can now merge P and $\langle \{\pi\} \rangle \mathbin{\backslash\!\backslash} X$ as follows. While decomposing π into disjoint cycles, we check for each $i \in \{1, \ldots, l\}$ and $j \in \{1, \ldots, p_i - 1\}$ whether the values at indices $x_{j,i}$ and $x_{j+1,i}$ of

$M(P)$ differ. If $M(P)[x_{j,i}] \neq M(P)[x_{j+1,i}]$ then $x_{j,i}$ and $x_{j+1,i}$ do not belong to the same cell of P. Hence we connect $x_{j,i}$ with $x_{j+1,i}$ in $F(P)$. As a result of these operations we obtain a disconnected graph $G(Q)$ with vertex set $X = \{1, \ldots, n\}$ such that each cell of Q corresponds to a unique *connected component* in $G(Q)$. Finally, we can transform $G(Q)$ into $F(Q)$, that is, removing redundant edges in the connected components in $G(Q)$, and meanwhile construct the array $M(Q)$ in time linear in $|X|$ using depth-first search on $G(Q)$.

Remark 5.3. A set-union algorithm [93, 104] provides an alternative for maintaining orbit partitions. •

5.3.4 Analysis and empirical data

A worst-case analysis of Algorithm 5.3 and 5.4 is certainly not straightforward. We restrict the analysis to the actual cost involved in checking both pruning criteria. The criterion outlined in Section 5.3.1 requires only a minor modification to the canonicity algorithm. Each call to the function *rightcoset* (❷ in Algorithm 5.3) results in an additional equality check. Therefore, even when the recursion tree is not additionally pruned, the application of the criterion itself does not affect the run-time of the canonicity algorithm.

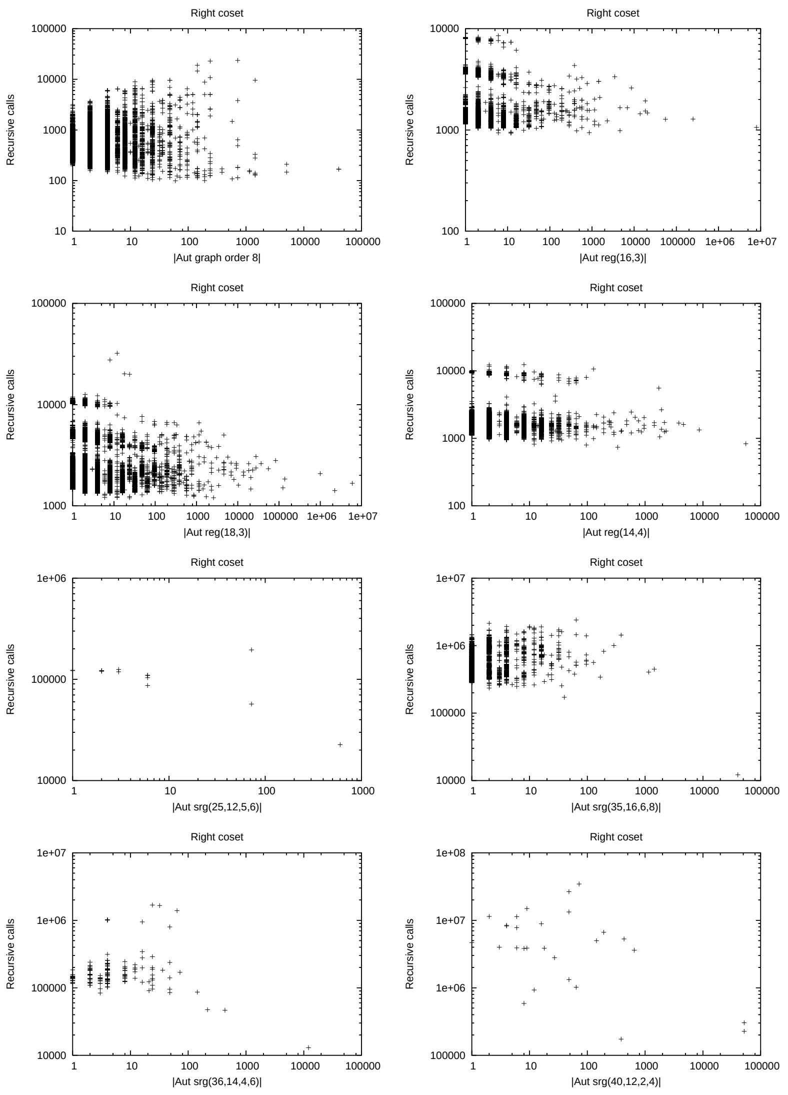
The pruning criterion outlined in Section 5.3.2 requires somewhat more modifications. When M is minimal, we now have to check exactly $\frac{1}{2}n(n-1)$ times whether a given element is the minimal element in its orbit under a group $\langle S \rangle$ where S is the set of automorphisms of M encountered so far (❶ in Algorithm 5.4). Each such operation takes constant-time and therefore has only a minimal impact on the run-time of the canonicity algorithm. Each time we discover an automorphism π, we have to update $\langle S \rangle \backslash\backslash X$ to $\langle S \cup \{\pi\} \rangle \backslash\backslash X$ where $X = \{1, \ldots, n\}$. Each update entails the merger of distinct orbits and thus the possibility to discard more right coset in the remainder of the traversal.

The effectiveness of both criteria is examined by comparing data obtained from the orderly generation of the same classes of graphs as in Section 5.2.3. In Table 5.2 and 5.3 Algorithm 5.2, 5.3 and 5.4 are compared. The total number of graphs along with the total number of permutations (partial or complete) which are checked during all executed canonicity tests is given. The empirical data illustrates that the degree of additional pruning differs considerably. Both pruning criteria turn out to be particularly effective when a substantial part of the (sub)matrices encountered, have a relatively large automorphism group. When both criteria are less effective, it turns out that the additional cost involved in checking both criteria has only a minor impact on the overall performance of the orderly generation algorithm itself.

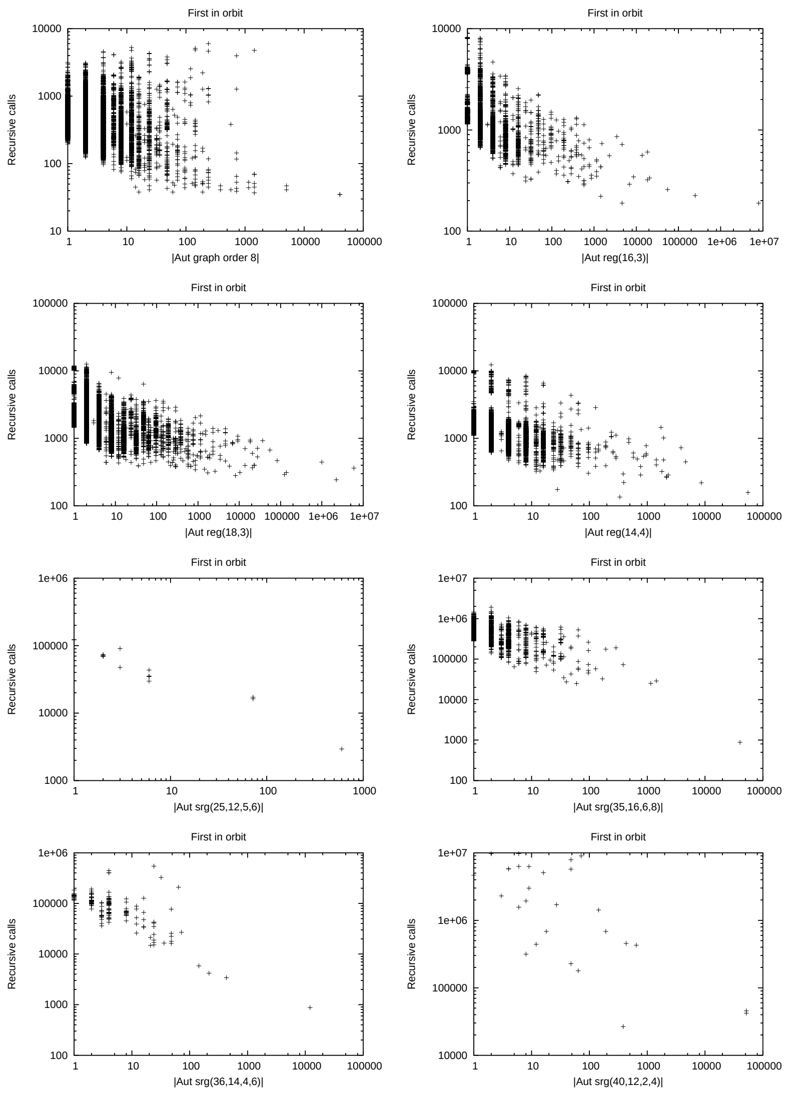
Finally we examine the effect of the size of the automorphism group of minimal representatives on the total number of permutations checked by both Algorithm 5.3 and 5.4. We consider canonicity tests on the same representatives as in Section 5.2.3. Figure 5.2 and 5.3 indicate that the number of nodes in the recursion tree tends to decrease as the size of the automorphism group increases. This correlation is even more explicit for Algorithm 5.4 than for Algorithm 5.3. This confirms the interpretation of the results in Table 5.2 and 5.3.

		Algorithm 5.2	Algorithm 5.3	Algorithm 5.4
v k	$\exists$	No. of perm.	No. of perm.	No. of perm.
8 –	12346	11 686 005	9 727 187	7 574 532
9 –	274668	414 180 087	377 494 839	327 260 260
9 4	16	93 267	71 628	51 393
9 6	4	52 154	28 715	11 564
10 3	21	209 220	133 861	97 183
10 4	60	740 636	535 224	425 935
11 4	266	5 123 710	4 450 581	3 847 835
11 6	266	11 825 305	10 818 023	9 346 277
12 3	94	3 415 505	2 086 198	1 670 123
12 4	1547	44 723 209	41 001 545	37 525 543
12 5	7849	280 177 784	266 973 129	254 373 579
13 4	10786	443 200 687	419 663 282	397 968 566
14 3	540	58 971 987	35 043 524	30 365 138
14 4	88193	4 842 350 124	4 668 546 811	4 527 310 381
15 4	805579	57 568 896 199	56 084 159 733	55 037 048 650
16 3	4207	2 307 251 665	646 566 172	592 273 253
18 3	42110	275 808 095 342	13 226 739 376	12 551 512 846

Tabel 5.2: Comparison of Algorithm 5.1, 5.2 and 5.3 applied in the orderly generation of simple graphs of order v and regular graphs of order v and degree k.



Figuur 5.2: Recursive calls of Algorithm 5.3 for classes of minimal representatives.



Figuur 5.3: Recursive calls of Algorithm 5.4 for classes of minimal representatives.

					Algorithm 5.2	Algorithm 5.3	Algorithm 5.4
v	k	λ	μ	$\exists$	No. of perm.	No. of perm.	No. of perm.
5	2	0	1	1	120	75	40
9	4	1	2	1	2 918	1 065	457
10	3	0	1	1	5 038	1 615	721
	6	3	4		8 658	3 720	1 214
13	6	2	3	1	19 331	9 836	5 643
15	6	1	3	1	80 932	23 395	7 906
	8	4	4		130 624	49 240	12 224
16	5	0	2	1	279 324	61 244	21 687
	10	6	6		416 214	178 510	36 090
16	6	2	2	2	199 408	53 820	23 507
	9	4	6		300 935	142 364	47 539
17	8	3	4	1	129 214	75 088	41 109
21	10	3	6	1	964 046	326 334	75 091
	10	5	4		2 095 791	696 204	144 207
25	8	3	2	1	10 595 498	1 303 854	458 845
	16	9	12		17 598 702	9 524 965	2 654 299
25	12	5	6	15	38 797 658	37 755 287	34 546 360
26	10	3	4	10	13 682 715	13 481 781	12 714 925
	15	8	9		282 392 851	279 477 345	257 672 546
27	10	1	5	1	26 170 822	5 242 960	1 289 795
	16	10	8		44 774 766	20 439 982	3 281 897
28	12	6	4	4	43 370 983	16 902 610	4 828 422
	15	6	10		23 207 607	13 230 035	4 068 141
29	14	6	7	41	9 317 917 605	3 315 667 998	3 291 720 746
36	10	4	2	1	934 984 410	85 113 426	23 754 480
	25	16	20		1 427 025 661	793 689 302	143 366 816
40	12	2	4	1	159 203 098 390	157 300 885 219	154 334 556 518
36	14	4	6	180	30 786 263 393	30 758 028 420	30 634 602 941
36	14	7	4	1	612 920 754	155 708 418	22 069 825
45	16	8	4	1	11 170 823 310	2 509 788 844	305 971 694
50	7	0	1	1	162 019 899 199	53 081 900 507	9 016 393 600

Tabel 5.3: Comparison of Algorithm 5.2, 5.3 and 5.4 applied in an orderly algorithm for strongly regular graphs

5.4 Image partitioning

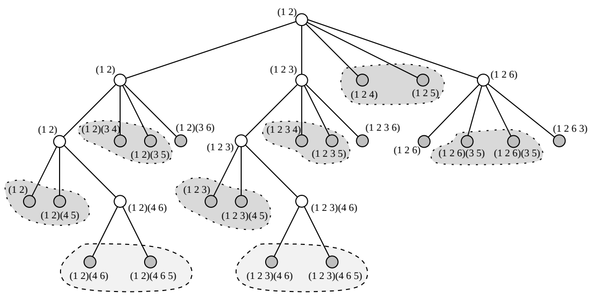
Up to now we have considered our column order canonicity algorithm for general matrices $M \in \mathcal{M}_n$. Since our canonicity algorithm is repeatedly applied during a single orderly generation, we can impose some additional restrictions on M. These restrictions will enable us to further optimize our canonicity algorithm. In this section we will make use of the knowledge that M must always be lexically ordered. After all, assume that M is not lexically ordered, then the (partially instantiated) matrix M would already have been discarded by the orderly generation algorithm before the actual execution of the canonicity test.

5.4.1 Drawbacks of the partial permutation criterion

The way in which we check the partial permutation criterion during traversal of $\mathrm{Sym}(n)$ is open to improvement. The manner in which we check this criterion causes the recursion tree to be pruned in different parts of the recursion for exactly the same reasons. Below we give a formal description of this drawback. First we shall illustrate this by means of an example.

Example 5.7. Consider $M \in \mathcal{M}_6$ below. Let $\sigma = (1\ 2)$. The recursion tree shown corresponds to the traversal of $U^{(1)}\sigma$ by Algorithm 5.4.

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 2 & 1 \\ 1 & 1 & 0 & 2 & 2 & 2 \\ 1 & 2 & 2 & 0 & 1 & 1 \\ 1 & 2 & 2 & 1 & 0 & 2 \\ 2 & 1 & 2 & 1 & 2 & 0 \end{pmatrix}$$



Note that

$$M_{1,5} = 1, \quad M_{1^g,4} = M_{1^\sigma,4} = 2 \quad \text{and} \quad M_{1^g,5} = M_{1^\sigma,5} = 2, \tag{5.12}$$

for each $g \in U^{(1)}\sigma$. Each right coset $U^{(5)}\pi \subset U^{(1)}\sigma$ with either $5^\pi = 4$

or $5^\pi = 5$ which is still to be considered at level 5 in the recursion can be discarded entirely. Indeed $(h\,M)_4 = M_4$ for each $h \in U^{(5)}\,\pi$; otherwise the recursion tree would have been pruned at a higher level. Using (5.12) we find that $(h\,M)_5 > M$ for each such h. This behaviour can be predicted even before we traverse $U^{(1)}\,\sigma$, after all, discarding $U^{(5)}\,\pi$ at level 5 in the recursion only depends on the value of 5^σ. Consider the following examples:

- Let $g_1 = (1\ 2)\ (4\ 6)$, $g_2 = (1\ 2\ 3)\ (4\ 6) \in U^{(1)}\,\sigma$. Note that $(g_1\,M)_4 = (g_2\,M)_4 = M_4$ and $5^{g_1} = 5^{g_2} = 5$, hence
$$(g_1\ M)_{1,5} = (g_2\ M)_{1,5} = M_{1^\sigma,5} > M_{1,5}.$$
As a result of this, $U^{(5)}\ g_1$ and $U^{(5)}\ g_2$ (light gray area) can both be discarded upon consideration at level 5 in the recursion.
- Let $h_1 = (1\ 2)\ (4\ 6\ 5)$, $h_2 = (1\ 2\ 3)\ (4\ 6\ 5) \in U^{(1)}\,\sigma$. Note that $(h_1\ M)_4 = (h_2\,M)_4 = M_4$ and $5^{h_1} = 5^{h_2} = 4$, hence
$$(h_1\ M)_{1,5} = (h_2\ M)_{1,5} = M_{1^\sigma,4} > M_{1,5}.$$
As a result of this, $U^{(5)}\ h_1$ and $U^{(5)}\ h_2$ (light gray area) can both be discarded upon consideration at level 5 in the recursion.

In both cases the partial permutation criterion prunes the recursion tree in different parts of the recursion for the same reason. This behaviour can be predicted higher up in the recursion, that is, before $U^{(1)}\,\sigma$ is traversed. Let $l \in \{2, \ldots, 4\}$. Clearly $M_{1,l} = 1$. We leave it for the reader to verify that similar behaviour can be observed for right cosets $U^{(l)}\,\tau$ with either $l^\tau = 4$ or $l^\tau = 5$ considered at level l (dark gray areas). •

In order to give a formal description of this phenomenon, we first need to define the concepts of *pivot level* and *error level*.

Definition 5.4.1. Let $M \in \mathcal{M}_n$ and $\pi \in \mathrm{Sym}(n)$ such that $\pi\,M \neq M$. Define the *error level* $\mathsf{err}^\pi(M)$ of π on M to be the smallest integer $l \in \{2, \ldots, n\}$ such that $(\pi\,M)_l \neq M_l$ and the corresponding *pivot level* $\mathsf{piv}^\pi(M)$ of π on M to be the smallest integer $q \in \{1, \ldots, l-1\}$ such that $(\pi\,M)_{1\ldots q,l} \neq M_{1\ldots q,l}$. ◇

Note that $l = \mathsf{err}^\pi(M)$ and $q = \mathsf{piv}^\pi(M)$ correspond to the first position where the certificates $\mathcal{C}(\pi\,M)$ and $\mathcal{C}(M)$ differ. It follows immediately that M can not be in canonical form when $(\pi M)_{q,l} < M_{q,l}$. For each $g \in U^{(l)}\,\pi$ we have $\mathsf{err}^g(M) = l$ and $\mathsf{piv}^g(M) = \mathsf{piv}^\pi(M)$ because π and g have the same l-prefix. For future notational simplicity we define $\mathsf{err}^\pi(M) = n+1$ and $\mathsf{piv}^\pi(M) = 0$ when $\pi M = M$.

The following theorem translates this phenomenon in terms of error level and pivot level.

Theorem 5.4.2. *Let $M \in \mathcal{M}_n$, $\sigma \in \mathrm{Sym}(n)$, $q \in \{1, \ldots, n-1\}$ and k, $l \in \{q+1, \ldots, n\}$ such that*

$$(\sigma M)_q = M_q, \quad (\sigma M)_{1\ldots q-1,k} = M_{1\ldots q-1,l} \quad \textit{and} \quad (\sigma M)_{q,k} \neq M_{q,l}. \tag{5.13}$$

Then for each $\pi \in U^{(q)}\,\sigma$ with $(\pi M)_{l-1} = M_{l-1}$ and $l^\pi = k^\sigma$ we have $\mathsf{err}^\pi(M) = l$ and $\mathsf{piv}^\pi(M) = q$. Moreover, we have $(\sigma M)_{q,k} > M_{q,l}$ if and only if $(\pi M)_l > M_l$ whereas $(\sigma M)_{q,k} < M_{q,l}$ if and only if $(\pi M)_l < M_l$.

Proof : Both σ and π have the same q-prefix. Hence

$$(\pi M)_{1\ldots q,l} = M_{1^\pi \ldots q^\pi, l^\pi} = M_{1^\sigma \ldots q^\sigma, l^\pi} = M_{1^\sigma \ldots q^\sigma, k^\sigma} = (\sigma M)_{1\ldots q,k} \neq M_{1\ldots q,l}$$

as $l^\pi = k^\sigma$. Moreover $(\pi M)_{l-1} = M_{l-1}$ and therefore $\mathsf{err}^\pi(M) = l$ and $\mathsf{piv}^\pi(M) = q$ by definition. ■

We reconsider Example 5.7 in these terms.

Example 5.8. Let M and $\sigma = (1\ 2)$ be as before. Theorem 5.4.2 states that $\mathsf{err}^\pi(M) = 5$ and $\mathsf{piv}^\pi(M) = 1$ for each $\pi \in U^{(1)}\,\sigma$ with either $5^\pi = 4$ or $5^\pi = 5$ such that $(\pi M)_4 = M_4$. E.g., reconsider $g_1 = (1\ 2)\ (4\ 6) \in U^{(1)}\,\sigma$. Note that $(g_1 M)_4$ and $5^{g_1} = 5$. Hence $(g_1 M)_{1,5} = M_{1^\sigma,5} > M_{1,5}$ and therefore $\mathsf{err}^{g_1}(M) = 5$ and $\mathsf{piv}^{g_1}(M) = 1$. Similarly for $l \in \{2, \ldots, 4\}$ we have $\mathsf{err}^\tau(M) = 5$ and $\mathsf{piv}^\tau(M) = 1$ for each $\tau \in U^{(1)}\,\sigma$ with either $l^\tau = 4$ or $l^\tau = 5$ such that $(\tau M)_{l-1} = M_{l-1}$. •

In general, let $M \in \mathcal{M}_n$ and $l \in \{2, \ldots, n\}$. Assume that we are at level l in the recursion of Algorithm 5.4 and are about to consider the right coset $U^{(l)}\,\pi$. Recall that at this point $(\pi M)_{l-1} = M_{l-1}$; otherwise the recursion tree would have been pruned at a higher level in the recursion. Before we actually traverse $U^{(l)}\,\pi$ recursively, the partial permutation criterion compares $(\pi M)_{1\ldots l-1,l}$

with $M_{1\ldots l-1,l}$. Based on this comparison the canonicity algorithm may either recurse, discard $U^{(l)}\,\pi$ entirely or even terminate. If $(\pi\,M)_{1\ldots l-1,l} = M_{1\ldots l-1,l}$, then the canonicity algorithm recurses by considering $U^{(l)}$ as a disjoint union of right cosets of $U^{(l+1)}$. If not, then $\mathsf{err}^{\pi}(M) = l$ and the course of the algorithm is determined by $(\pi\,M)_{q,l}$ where $q = \mathsf{piv}^{\pi}(M)$. Now let σ be the representative of the right coset of $U^{(q)}$ considered previously at level q in the recursion. Clearly $\pi \in U^{(q)}\,\sigma$. Hence, as π and σ have the same q-prefix, we find that $(\sigma\,M)_q = M_q$ and $(\sigma\,M)_{1\ldots q,k} = (\pi\,M)_{1\ldots q,l}$ for some $k \in \{q+1,\ldots,n\}$ such that $k^{\sigma} = l^{\pi}$. We distinguish between the following cases.

1. If $(\sigma\,M)_{q,k} = (\pi\,M)_{q,l} < M_{q,l}$, then according to Theorem 5.4.2 we find that at recursion level q, that is, when the q-prefix of π is known and in particular the image of q is fixed, we may already predict for π the action to be taken at a deeper level l in the recursion, that is, terminating the canonicity test.

2. Similarly, if $(\sigma\,M)_{q,k} = (\pi\,M)_{q,l} > M_{q,l}$, then at the same recursion level q we may already predict for π the action to be taken at a deeper level l in the recursion, that is, discarding $U^{(l)}\,\pi$ entirely.

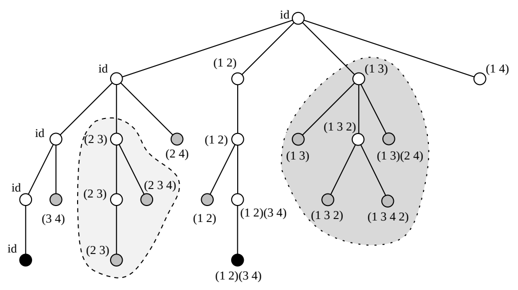
Within this context the error level $\mathsf{err}^{\pi}(M)$ corresponds to the level in the recursion at which the recursion tree is actually pruned, whereas the pivot level $\mathsf{piv}^{\pi}(M)$ identifies the level in the recursion at which this pruning can be predicted. Not only for π we can predict the action to be taken. Now let $\pi' \in U^{(q)}\,\sigma$ satisfy $(\pi'\,M)_{l-1} = M_{l-1}$. If also $l^{\pi'} = l^{\pi}$, then $\mathsf{err}^{\pi'}(M) = \mathsf{err}^{\pi}(M)$ and $\mathsf{piv}^{\pi'}(M) = \mathsf{piv}^{\pi}(M)$. After all π' has the same q-prefix as π. Hence the action to be taken at level l in the recursion will be the same for π' as it is for π.

More generally, we can predict for each $g \in U^{(q)}\,\sigma$ with $l^{g} = l^{\pi}$ (among which π and π') we encounter at a deeper level l in the recursion the action to be taken. Naturally, if the action to be taken is to terminate then only the first such g we encounter causes the canonicity algorithm to terminate. The partial permutation criterion clearly tends to prune the recursion tree in different parts of the recursion for exactly the same reason, what's more, a reason which depends only on l^{g} and which can be determined higher up, that is, at pivot level $\mathsf{piv}^{\pi}(M)$ in the recursion. This repeated pruning for the same reason has a negative impact on the performance of the criterion. In the next sections we will improve the way in which the criterion is checked by using the information already available at pivot level $\mathsf{piv}^{\pi}(M)$.

Below we give some examples to illustrate the points made in the previous discussion.

Example 5.9. Consider $M \in \mathcal{M}_4$ below. The recursion tree shown corresponds to the traversal of $\mathrm{Sym}(4)$ by Algorithm 5.4.

$$M = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}$$



Let $\tau = (2\,3)$. Consider the traversal of $U^{(2)}\,\tau$ (light gray area). The right coset $U^{(4)}\,\tau$ is discarded because

$$(\tau\, M)_4 = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 2 & \mathbf{2} \\ 1 & 2 & 0 & 1 \\ 2 & 2 & 1 & 0 \end{pmatrix} > M_4.$$

Clearly $\mathsf{err}^{\tau}(M) = 4$ and $\mathsf{piv}^{\tau}(M) = 2$. The fact that $(\tau\, M)_{1\ldots 2,4} > M_{1\ldots 2,4}$ is established before the traversal of $U^{(2)}\,\tau$ allows us to predict the discarding of $U^{(4)}\,\tau$.

Let $\sigma = (1\,3)$. Consider the traversal of $U^{(1)}\,\sigma$ (dark gray area). Let $\rho = (1\,3\,2)$. The right coset $U^{(3)}\,\rho$ is discarded because

$$(\rho\, M)_3 = \begin{pmatrix} 0 & 1 & \mathbf{2} \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} > M_3.$$

Clearly $\mathsf{err}^{\rho}(M) = 3$ and $\mathsf{piv}^{\rho}(M) = 1$. The fact that $(\sigma\, M)_{1,2} > M_{1,3}$ is established before the traversal of $U^{(1)}\,\sigma$ allows us to predict the discarding of $U^{(3)}\,\rho$.

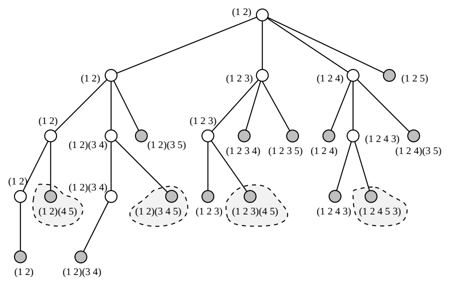
Let $\psi = (1\,3\,4\,2)$. The right coset $U^{(3)}\,\psi$ is discarded because

$$(\psi\, M)_3 = \begin{pmatrix} 0 & 1 & \mathbf{2} \\ 1 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix} > M_3.$$

Clearly $\mathsf{err}^{\psi}(M) = 3$ and $\mathsf{piv}^{\psi}(M) = 1$. The fact that $(\sigma\, M)_{1,4} > M_{1,3}$ is established before the traversal of $U^{(1)}\,\sigma$ allows us to predict the discarding of $U^{(3)}\,\psi$. Note that the actual reason of discarding $U^{(3)}\,\psi$ and $U^{(3)}\,\psi$ is not the same as $4^{\psi} \neq 4^{\rho}$. •

Example 5.10. Consider $M \in \mathcal{M}_5$ below. Let $\sigma = (1\ 2)$. The recursion tree shown corresponds to the traversal of $U^{(1)}\,\sigma$ by Algorithm 5.4.

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 2 \\ 1 & 1 & 0 & 2 & 1 \\ 1 & 1 & 2 & 0 & 2 \\ 1 & 2 & 1 & 2 & 0 \end{pmatrix}$$



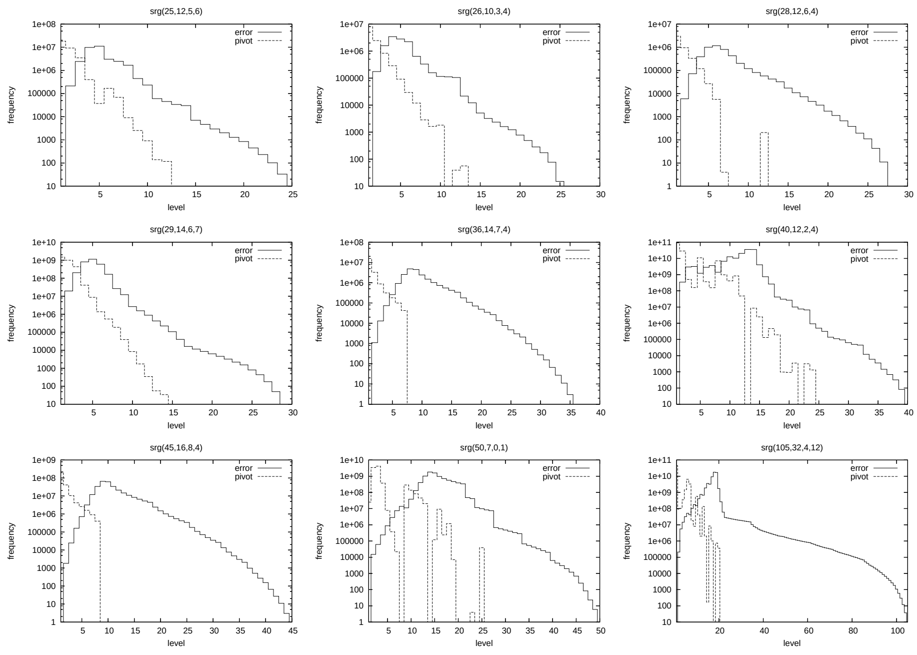
Let $\pi_1 = (1\ 2)\ (4\ 5)$, $\pi_2 = (1\ 2)\ (3\ 4\ 5)$, $\pi_3 = (1\ 2\ 3)(4\ 5)$, $\pi_4 = (1\ 2\ 4\ 5\ 3)$ and $i \in \{1, \ldots, 4\}$. Each corresponding right cosets $U^{(4)}\,\pi_i$ is discarded (light gray areas) because

$$(\pi_1\, M)_4 = \begin{pmatrix} 0 & 1 & 1 & \mathbf{2} \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{pmatrix} > M_4 \qquad (\pi_2\, M)_4 = \begin{pmatrix} 0 & 1 & 1 & \mathbf{2} \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix} > M_4$$

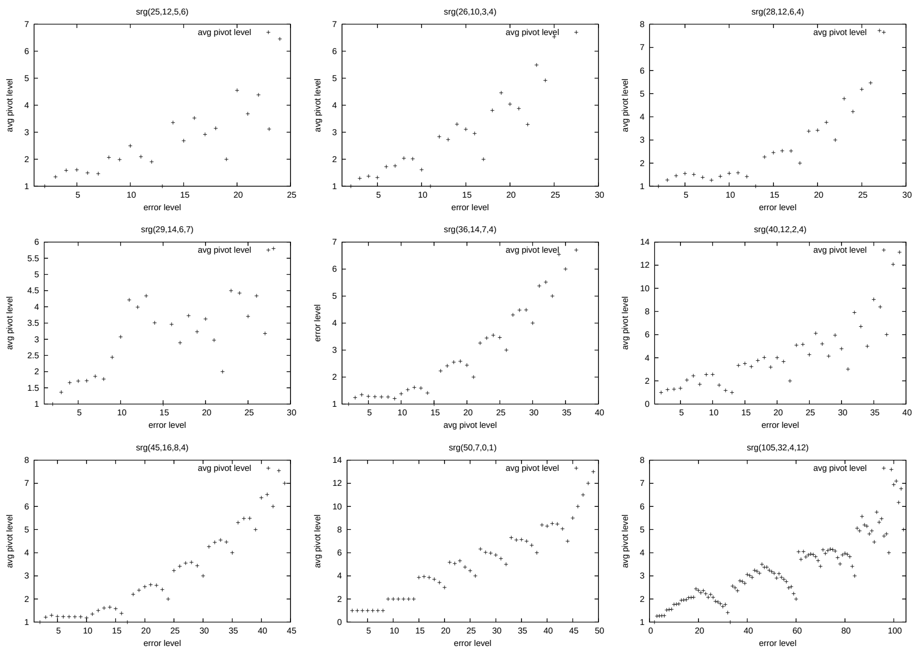
$$(\pi_3\, M)_4 = \begin{pmatrix} 0 & 1 & 1 & \mathbf{2} \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{pmatrix} > M_4 \qquad (\pi_4\, M)_4 = \begin{pmatrix} 0 & 1 & 1 & \mathbf{2} \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 2 & 2 & 1 & 0 \end{pmatrix} > M_4.$$

Clearly $\mathsf{err}^{\pi_i}(M) = 4$ and $\mathsf{err}^{\pi_i}(M) = 1$ for each such π_i. The fact that $(\sigma\, M)_{1,5} > M_{1,4}$ is established prior to the traversal of $U^{(1)}\,\sigma$ allows us to predict the discarding of these cosets. Note that the actual reason of discarding is the same as $4^{\pi_i} = 5$ for each such π_i. •

One can expect this behaviour to occur more frequently when the difference between error levels and the corresponding pivot levels is rather large. Empirical data obtained from the orderly generation of strongly regular graphs shows that large differences do indeed arise. In Figure 5.4 we represent the frequency of occurrence of error and pivot levels among all canonicity tests executed during orderly generation, whereas in Figure 5.5 we represent for each such error level the average corresponding pivot level. For each parameter set we can observe a relatively large distance between the error levels and the corresponding average pivot levels. Moreover, most pivot levels are highly concentrated at the more shallow levels in the recursion. Both observations affirm that altering the way in which we check the partial permutation criterion could improve its performance.



Figuur 5.4: Frequency of pivot and error levels.



Figuur 5.5: The average pivot level per error level.

5.4.2 Canonicity algorithm with image partitioning

The discussion in the above section shows that the manner in which we check the partial permutation criterion is open to improvement. In subsequent sections we give a top-down description of the actual modifications to the canonicity algorithm. The current section introduces a framework which serves for describing these modification at a high conceptual level. A somewhat more detailed, low-level description is deferred until the next section.

The improvement to our canonicity algorithm will allow us to further restrict the number of right cosets we have to consider. In order to do so we need to introduce an ordering on the set of images of the permutations involved. We use the lexicographic ordering $\leq$ on tuples to define the binary relation $\overset{i,\pi,M}{\lesssim}$ on the set $\{1,\ldots,n\}$, as follows:

Definition 5.4.3. Let $M \in \mathcal{M}_n$, $\pi \in \mathrm{Sym}(n)$ and $i, p, q \in \{1,\ldots,n\}$. Then $p \overset{i,\pi,M}{\lesssim} q$ if and only if $M_{1^\pi \ldots i^\pi, p} \leq M_{1^\pi \ldots i^\pi, q}$. ◇

If $\pi = \mathrm{id}$ then we write $\overset{i,M}{\lesssim}$ instead of $\overset{i,\mathrm{id},M}{\lesssim}$. If π and π' have the same i-prefix, then $\overset{i,\pi,M}{\lesssim}$ is equivalent to $\overset{i,\pi',M}{\lesssim}$. The relation $\overset{i,\pi,M}{\lesssim}$ is a *weak order* since it is both *complete* and *transitive*. Therefore $\overset{i,\pi,M}{\sim}$ is an equivalence relation on $\{1,\ldots,n\}$.

Definition 5.4.4. Let $M \in \mathcal{M}_n$, $\pi \in \mathrm{Sym}(n)$ and $i, p, q \in \{1,\ldots,n\}$. Then $p \overset{i,\pi,M}{\sim} q$ if and only if $p \overset{i,\pi,M}{\lesssim} q$ and $q \overset{i,\pi,M}{\lesssim} p$. ◇

As before we write $\overset{i,M}{\sim}$ instead of $\overset{i,\mathrm{id},M}{\lesssim}$ and again $\overset{i,\pi,M}{\lesssim}$ only depends on the i-prefix of π. We denote the equivalence class of $p \in \{1,\ldots,n\}$ by $[p]_{i,\pi,M}$. Since only the diagonal entries of M are zero, it follows immediately that $[p^\pi]_{i,\pi,M} = \{p^\pi\}$ when $p \leq i$. The weak order $\overset{i,\pi,M}{\lesssim}$ induces a *total order* $\overset{i,\pi,M}{\leq}$ on the set of equivalence classes of $\overset{i,\pi,M}{\sim}$, as follows:

Definition 5.4.5. Let $M \in \mathcal{M}_n$, $\pi \in \mathrm{Sym}(n)$ and $i, p, q \in \{1,\ldots,n\}$. Then $[p]_{i,\pi,M} \overset{i,\pi,M}{\leq} [q]_{i,\pi,M}$ if and only if $p \overset{i,\pi,M}{\lesssim} q$. ◇

Again we write $\overset{i,M}{\leq}$ instead of $\overset{i,\pi,M}{\leq}$. Note that $\overset{i,\pi,M}{\leq}$ is *antisymmetric* while $\overset{i,\pi,M}{\lesssim}$ not necessarily. Recall from set theory that weak orders are in one-to-one correspondence with ordered partitions on the same set.

An *ordered partition* P of a set V is a sequence $P = (V_1, V_2, \dots, V_r)$ of disjoint non-empty subsets of V whose union is V. The number r of subsets in P shall be denoted by $|P|$. The set of all ordered partitions of V shall be denoted by $\mathcal{P}(V)$. Usually the elements of an ordered partition $P \in \mathcal{P}(V)$ are called its *cells*. The a-th cell of P is denoted by $P_{[a]}$. A *trivial* cell of P is a cell of cardinality one. If every cell of P is trivial, then P is called a *discrete* ordered partition. P is the *unit* partition on V if $P = (V)$.

Let $P \in \mathcal{P}(V)$ and $x \in V$, then we define $\mathsf{cell}(P, x)$ to be the index i of the unique cell $P_{[i]}$ to which x belongs. Let $P', P'' \in \mathcal{P}(V)$, then P' is a *refinement* of P'', denoted by $P' \leq P''$, if every cell of P' is a subset of some cell of P''. Moreover, P' is a *cell order preserving refinement* of P'', denoted by $P' \preceq P''$, if $P' \leq P''$ and $\mathsf{cell}(P', x) < \mathsf{cell}(P', y)$ implies $\mathsf{cell}(P'', x) \leq \mathsf{cell}(P'', y)$ for all $x, y \in V$. For example, $(\{1\}, \{3\}, \{2\}) \leq (\{1, 2\}, \{3\})$, $(\{1\}, \{3\}, \{2\}) \not\preceq (\{1, 2\}, \{3\})$ and $(\{2\}, \{1\}, \{3\}) \preceq (\{1, 2\}, \{3\})$

Definition 5.4.6. Let $M \in \mathcal{M}_n$, $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \dots, n-1\}$. Define $\mathsf{lex}_i^\pi(M)$ to be the ordered partition on $\{(i+1)^\pi, \dots, n^\pi\}$ that uniquely corresponds to the weak order $\overset{i,\pi,M}{\lesssim}$. ◇

The cells of $\mathsf{lex}_i^\pi(M)$ are the equivalence classes of $\overset{i,\pi,M}{\approx}$ on $\{(i+1)^\pi, \dots, n^\pi\}$ and $\mathsf{lex}_i^\pi(M)_{[a]} \overset{i,\pi,M}{\leq} \mathsf{lex}_i^\pi(M)_{[b]}$ if and only if $a \leq b$. Note that we do not consider ordered partitions on the entire set $\{1, \dots, n\}$. In other words, its cells are the equivalence classes of $\overset{i,\pi,M}{\approx}$ on $\{1, \dots, n\}$, except that the equivalence classes $[p^\pi]_{i,\pi,M} = \{p^\pi\}$ with $p \leq i$ have been omitted. Note that since the cells of $\mathsf{lex}_i^\pi(M)$ are the equivalence classes of $\overset{i,\pi,M}{\approx}$, we find that the tuple $M_{1^\pi \dots, i^\pi, p}$ with $p \in \mathsf{lex}_i^\pi(M)_{[a]}$ is independent of the choice of p. As before we write $\mathsf{lex}_i(M)$ instead of $\mathsf{lex}_i^{\mathrm{id}}(M)$ and again $\mathsf{lex}_i^\pi(M)$ only depends on the i-prefix of π. We say that $\mathsf{lex}_i^\pi(M)$ is the *lexically ordered image partition* of M defined by the prefix of length i of π while $\mathsf{lex}_i(M)$ is the *lexically ordered partition* of the first i rows of M. For future notational simplicity we set $\mathsf{lex}_0^\pi(M) = \mathsf{lex}_0(M) = (\{1, \dots, n\})$, the *unit partition*.

Example 5.11. Let $\pi = (1\,2\,3\,5)$. Consider $M \in \mathcal{M}_5$ below, then $\mathsf{lex}_i(M)$ and

$\mathsf{lex}_i^\pi(M)$ for $i \in \{1,\dots,4\}$ are as listed.

$$M = \left(\begin{array}{ccccc} 0 & \vline\; 1 & 1 & 1 & 1 \\ 1 & 0 & \vline\; 1 & 1 & \vline\; 2 \\ 1 & 1 & 0 & \vline\; 2 & \vline\; 1 \\ 1 & 1 & 2 & 0 & \vline\; 2 \\ 1 & 2 & 1 & 2 & 0 \end{array}\right) \quad \begin{array}{ll} \mathsf{lex}_0(M) = (\{1,2,3,4,5\}) & \mathsf{lex}_0^\pi(M) = (\{1,2,3,4,5\}) \\ \mathsf{lex}_1(M) = (\{2,3,4,5\}) & \mathsf{lex}_1^\pi(M) = (\{1,3,4\},\{5\}) \\ \mathsf{lex}_2(M) = (\{3,4\},\{5\}) & \mathsf{lex}_2^\pi(M) = (\{1\},\{4\},\{5\}) \\ \mathsf{lex}_3(M) = (\{4\},\{5\}) & \mathsf{lex}_3^\pi(M) = (\{1\},\{4\}) \\ \mathsf{lex}_4(M) = (\{5\}) & \mathsf{lex}_4^\pi(M) = (\{1\}) \end{array}$$

•

Example 5.12. Let $\pi = (1\,3)\,(2\,4)$. Consider $M \in \mathcal{M}_5$ below, then $\mathsf{lex}_i(M)$ and $\mathsf{lex}_i^\pi(M)$ for $i \in \{0,\dots,4\}$ are as listed.

$$M = \left(\begin{array}{ccccc} 0 & \vline\; 1 & \vline\; 2 & 2 & 2 \\ 1 & 0 & \vline\; 2 & 2 & 2 \\ 2 & 2 & 0 & \vline\; 1 & 1 \\ 2 & 2 & 1 & 0 & \vline\; 2 \\ 2 & 2 & 1 & 2 & 0 \end{array}\right) \quad \begin{array}{ll} \mathsf{lex}_0(M) = (\{1,2,3,4,5\}) & \mathsf{lex}_0^\pi(M) = (\{1,2,3,4,5\}) \\ \mathsf{lex}_1(M) = (\{2\},\{3,4,5\}) & \mathsf{lex}_1^\pi(M) = (\{4,5\},\{1,2\}) \\ \mathsf{lex}_2(M) = (\{3,4,5\}) & \mathsf{lex}_2^\pi(M) = (\{5\},\{1,2\}) \\ \mathsf{lex}_3(M) = (\{4,5\}) & \mathsf{lex}_3^\pi(M) = (\{5\},\{2\}) \\ \mathsf{lex}_4(M) = (\{5\}) & \mathsf{lex}_4^\pi(M) = (\{5\}) \end{array}$$

•

We now define the sequence $\mathsf{tu}_i^\pi(M)$ of i-tuples.

Definition 5.4.7. Let $M \in \mathcal{M}_n$, $\pi \in \mathrm{Sym}(n)$ and $i \in \{1,\dots,n-1\}$. Define

$$\mathsf{tu}_i^\pi(M) = \left(M_{1^\pi \dots i^\pi, j_1}, M_{1^\pi \dots i^\pi, j_2}, \dots, M_{1^\pi \dots i^\pi, j_{n-i}}\right)$$

such that $\{j_1,\dots,j_{n-i}\} = \{(i+1)^\pi,\dots,n^\pi\}$ and such that $j_a \le j_b$ if and only if $M_{1^\pi \dots i^\pi, j_a} \le M_{1^\pi \dots i^\pi, j_b}$, or equivalently if and only if $j_a \overset{i,\pi,M}{\lesssim} j_b$. ◇

We denote the a-th entry $M_{1^\pi \dots i^\pi, j_a}$ of $\mathsf{tu}_i^\pi(M)$ by $\mathsf{tu}_i^\pi(M)_a$. As before we write $\mathsf{tu}_i(M)$ instead of $\mathsf{tu}_i^{\mathrm{id}}(M)$ and again $\mathsf{tu}_i^\pi(M)$ only depends on the i-prefix of π. We say that $\mathsf{tu}_i^\pi(M)$ is the sequence of *lexically ordered image tuples* defined by the prefix of length i of π, while $\mathsf{tu}_i(M)$ is the sequence of *lexically ordered tuples* of the first i rows of M.

To put it differently, $\mathsf{tu}_i^\pi(M)$ is the sequence of tuples $M_{1^\pi \dots i^\pi, j}$ with $j \in \{(i+1)^\pi,\dots,n^\pi\}$, sorted according to the lexicographic ordering on tuples. This means that $\mathsf{tu}_i^\pi(M)$ can be uniquely written as a sequence of i-tuples

$$\mathsf{tu}_i^\pi(M) = (u_1,\dots,u_1, u_2,\dots,u_2, \dots, u_{r-1},\dots,u_{r-1}, u_r,\dots,u_r) \tag{5.14}$$

satisfying $u_a < u_b$ if and only if $a < b$. We write $\mathsf{tu}_i^{\pi}(M)_{[a]}$ for u_a, that is, the a-th unique tuple of $\mathsf{tu}_i^{\pi}(M)$. From the above we find that there is a strong relation between $\mathsf{tu}_i^{\pi}(M)$ and $\mathsf{lex}_i^{\pi}(M)$, as illustrated by the following theorem.

Theorem 5.4.8. *Let $M \in \mathcal{M}_n$, $\pi \in \mathrm{Sym}(n)$, $i \in \{1, \dots, n-1\}$. Then $\mathsf{tu}_i^{\pi}(M)_{[a]}$ is well-defined if and only if $a \in \{1, \dots, |\mathsf{lex}_i^{\pi}(M)|\}$ and then $\mathsf{tu}_i^{\pi}(M)_{[a]} = M_{1^\pi,\dots,i^\pi,p}$ for each $p \in \mathsf{lex}_i^{\pi}(M)_{[a]}$.*

In other words, with each cell $\mathsf{lex}_i^{\pi}(M)_{[a]}$ we can associate a unique tuple

$$\mathsf{tu}_i^{\pi}(M)_{[a]} = M_{1^\pi \dots, i^\pi, p}$$

which is independent of the choice of $p \in \mathsf{lex}_i^{\pi}(M)_{[a]}$. Moreover, it must hold that $\mathsf{tu}_i^{\pi}(M)_{[a]} \leq \mathsf{tu}_i^{\pi}(M)_{[b]}$ if and only if $a \leq b$.

Example 5.13. Let π and M as in Example 5.11, then

$$\begin{array}{rclrcl}
\mathsf{tu}_1(M) & = & ((1),(1),(1),(1)) & \mathsf{tu}_2(M) & = & ((1,1),(1,1),(1,2)) \\
\mathsf{tu}_1^{\pi}(M) & = & ((1),(1),(1),(2)) & \mathsf{tu}_2^{\pi}(M) & = & ((1,1),(1,2),(2,1))
\end{array}$$

•

Example 5.14. Let π and M as in Example 5.12, then

$$\begin{array}{rclrcl}
\mathsf{tu}_1(M) & = & ((1),(2),(2),(2)) & \mathsf{tu}_2(M) & = & ((2,2),(2,2),(2,2)) \\
\mathsf{tu}_1^{\pi}(M) & = & ((1),(1),(2),(2)) & \mathsf{tu}_2^{\pi}(M) & = & ((1,2),(2,2),(2,2))
\end{array}$$

•

The concept of lexically ordered image partitions allows us to outline the central idea behind the improvements to the canonicity algorithm. In general, let $M \in \mathcal{M}_n$ and $l \in \{2, \dots, n\}$. Assume we are at level $l-1$ in the recursion and Algorithm 5.4 considers the right coset $U^{(l-1)}\,\varphi$ satisfying $(\varphi\, M)_{l-1} = M_{l-1}$. We then successively traverse

$$U^{(l)}\,\varphi,\ U^{(l)}\,(l\;l+1)\,\varphi,\ \dots,\ U^{(l)}\,(l\;n-1)\,\varphi,\ U^{(l)}\,(l\;n)\,\varphi, \tag{5.15}$$

or to put it differently, successively extend the $(l-1)$-prefix of φ by each element of $\{l^\varphi, \dots, n^\varphi\}$ in turn. Note that if we set the image of l to be $q \in$

$\{l^\varphi, \dots, n^\varphi\}$, or equivalently consider the right coset $U^{(l)}\,(l\ q^{\varphi^{-1}})\,\varphi$, then for each $h \in U^{(l)}\,(l\ q^{\varphi^{-1}})\,\varphi$ we find that $(h\,M)_l$ can be written as

$$(h\,M)_l = \begin{pmatrix} M_{l-1} & v_q^T \\ v_q & 0 \end{pmatrix} \tag{5.16}$$

where in this context v_q serves as an abbreviation for $M_{1^\varphi \dots (l-1)^\varphi, q}$. After all, $(h\,M)_{l-1} = (\varphi\,M)_{l-1} = M_{l-1}$ and $M_{1^h \dots (l-1)^h, q} = M_{1^\varphi \dots (l-1)^\varphi, q}$ because φ and h have the same $(l-1)$-prefix. Moreover $v_q = \mathsf{tu}^\varphi_{l-1}(M)_{[a]}$ with $a = \mathsf{cell}(\mathsf{lex}^\varphi_{l-1}(M), q)$.

We can use $\mathsf{lex}^\varphi_{l-1}(M)$ to restrict the number of cosets in (5.15) we have to consider. The first cell of $\mathsf{lex}^\varphi_{l-1}(M)$, that is the subset of $\{l^\varphi, \dots, n^\varphi\}$ corresponding to the lexicographically smallest tuple $\mathsf{tu}^\varphi_{l-1}(M)_{[1]}$, will allow us to either discard each right coset in (5.15) or to consider only the right cosets $U^{(l)}\,(l\ q^{\varphi^{-1}})\,\varphi$ where $q \in \mathsf{lex}^\varphi_{l-1}(M)_{[1]}$. Let $u = M_{1 \dots l-1, l}$, then depending on the relation between u and $\mathsf{tu}^\varphi_{l-1}(M)_{[1]}$, that is, the l-th upperdiagonal column of respectively M and $(l\ p^{\varphi^{-1}})\,\varphi\,M$ with $p \in \mathsf{lex}^\varphi_l(M)_{[1]}$, we can decide the action to be taken. We distinguish between the following three cases:

1. If $u < \mathsf{tu}^\varphi_{l-1}(M)_{[1]}$, then we may discard $U^{(l-1)}\,\varphi$ entirely. Since $u < \mathsf{tu}^\varphi_{l-1}(M)_{[1]} \le \mathsf{tu}^\varphi_{l-1}(M)_{[a]}$ for each $a \in \{1, \dots, |\mathsf{lex}^\varphi_{l-1}(M)|\}$ we have $u < M_{1^\varphi \dots (l-1)^\varphi, r}$ for each $r \in \{l^\varphi, \dots, n^\varphi\}$. Therefore we find that

$$(h\,M)_l = \begin{pmatrix} M_{l-1} & v_r^T \\ v_r & 0 \end{pmatrix} > \begin{pmatrix} M_{l-1} & u^T \\ u & 0 \end{pmatrix} = M_l$$

   for each $h \in U^{(l)}\,(l\ r^{\varphi^{-1}})\,\varphi$. Hence no permutation of $U^{(l-1)}\,\varphi$ could lead to a counterexample to the minimality of M.

2. If $u > \mathsf{tu}^\varphi_{l-1}(M)_{[1]}$, then we may terminate the canonicity algorithm. Let $h \in U^{(l-1)}\,\varphi$ such that $l^h = p \in \mathsf{lex}^\varphi_{l-1}(M)_{[1]}$, then

$$(h\,M)_l = \begin{pmatrix} M_{l-1} & v_p^T \\ v_p & 0 \end{pmatrix} < \begin{pmatrix} M_{l-1} & u^T \\ u & 0 \end{pmatrix} = M_l.$$

   Hence h provides a counterexample to the minimality of M.

3. If $u = \mathsf{tu}^{\varphi}_{l-1}(M)_{[1]}$, then we may discard the right cosets $U^{(l)}\,(l\ r^{\varphi^{-1}})\,\varphi$ with $r \in \{l^{\varphi}, \ldots, n^{\varphi}\} \setminus \mathsf{lex}^{\varphi}_{l-1}(M)_{[1]}$ in (5.15) entirely. Because $u = \mathsf{tu}^{\varphi}_{l-1}(M)_{[1]} < \mathsf{tu}^{\varphi}_{l-1}(M)_{[b]}$ for each $b \in \{2, \ldots, |\mathsf{lex}^{\varphi}_{l-1}(M)|\}$ we have $u < M_{1^{\varphi}\ldots(l-1)^{\varphi},r}$ for each $r \in \{l^{\varphi}, \ldots, n^{\varphi}\} \setminus \mathsf{lex}^{\varphi}_{l-1}(M)_{[1]}$. Therefore we find that

$$(h\,M)_l = \begin{pmatrix} M_{l-1} & v_r^T \\ v_r & 0 \end{pmatrix} > \begin{pmatrix} M_{l-1} & u^T \\ u & 0 \end{pmatrix} = M_l$$

for each $h \in U^{(l)}\,(l\ r^{\varphi^{-1}})\,\varphi$. Therefore no permutation of $U^{(l)}\,(l\ r^{\varphi^{-1}})\,\varphi$ could ever lead to a counterexample to the minimality of M.

However we still have to consider the right cosets $U^{(l)}\,(l\ q^{\varphi^{-1}})\,\varphi$ with $q \in \mathsf{lex}^{\varphi}_{l-1}(M)_{[1]}$ in (5.15). Because $u = \mathsf{tu}^{\varphi}_{l-1}(M)_{[1]}$ we have $u = M_{1^{\varphi}\ldots(l-1)^{\varphi},q}$ for each $q \in \mathsf{lex}^{\varphi}_{l-1}(M)_{[1]}$. Therefore we find that

$$(h\,M)_l = \begin{pmatrix} M_{l-1} & v_q^T \\ v_q & 0 \end{pmatrix} = \begin{pmatrix} M_{l-1} & u^T \\ u & 0 \end{pmatrix} = M_l$$

for each $h \in U^{(l)}\,(l\ q^{\varphi^{-1}})\,\varphi$. Hence each permutation of $U^{(l)}\,(l\ q^{\varphi^{-1}})\,\varphi$ could still lead to a counterexample of the minimality of M.

Below we present some examples to illustrate the points made in the above discussion.

Example 5.15. Let π and M as in Example 5.11. Recall that $\mathsf{lex}^{\pi}_2(M) = (\{1\}, \{4\}, \{5\})$. Clearly $(\pi\,M)_2 = M_2$ and $M_{1\ldots2,3} = \mathsf{tu}^{\pi}_2(M)_{[1]}$. Write $u = M_{1\ldots2,3} = (1\ 1)$ and $z = \mathsf{tu}^{\pi}_2(M)_{[1]} = (1\ 1)$, then

$$(\varphi\,M)_3 = \begin{pmatrix} M_2 & z^T \\ z & 0 \end{pmatrix} = \begin{pmatrix} M_2 & u^T \\ u & 0 \end{pmatrix} = M_3$$

for each $\varphi \in U^{(3)}\,(3\ q^{\pi^{-1}})\,\pi$ with $q \in \mathsf{lex}^{\pi}_2(M)_{[1]} = \{1\}$. Moreover, for each $\rho \in U^{(3)}\,(3\ r^{\pi^{-1}})\,\pi$ and $\sigma \in U^{(3)}\,(3\ s^{\pi^{-1}})\,\pi$ with respectively $r \in \mathsf{lex}^{\pi}_2(M)_{[2]} = \{4\}$ and $s \in \mathsf{lex}^{\pi}_2(M)_{[3]} = \{5\}$ we have

$$(\rho\,M)_3 = \begin{pmatrix} M_2 & v^T \\ v & 0 \end{pmatrix} > \begin{pmatrix} M_2 & u^T \\ u & 0 \end{pmatrix} = M_3$$

$$(\sigma M)_3 = \begin{pmatrix} M_2 & w^T \\ w & 0 \end{pmatrix} > \begin{pmatrix} M_2 & u^T \\ u & 0 \end{pmatrix} = M_3$$

where $v = \mathsf{tu}_2^\pi(M)_{[2]} = (1\ 2)$ and $w = \mathsf{tu}_2^\pi(M)_{[3]} = (2\ 1)$. •

Example 5.16. Let π and M as in Example 5.12. Recall that $\mathsf{lex}_2^\pi(M) = (\{5\}, \{1,2\})$. Clearly $(\pi M)_2 = M_2$ and $M_{1\ldots 2,3} < \mathsf{tu}_2^\pi(M)_{[1]}$. Write $u = M_{1\ldots 2,3} = (2\ 2)$ and $v = \mathsf{tu}_2^\pi(M)_{[1]} = (1\ 2)$, then

$$(\varphi M)_3 = \begin{pmatrix} M_2 & v^T \\ v & 0 \end{pmatrix} < \begin{pmatrix} M_2 & u^T \\ u & 0 \end{pmatrix} = M_3$$

for each $\varphi \in U^{(3)}\,(3\; q^{\pi^{-1}})\,\pi$ with $q \in \mathsf{lex}_2^\pi(M)_{[1]} = \{5\}$. •

If we would like to reduce the number of right cosets in (5.15) we have to consider, using the strategy above, then we need to compute $\mathsf{lex}_{l-1}^\varphi(M)$ in some way or other. This essentially amounts to sorting the set $\{l^\varphi, \ldots, n^\varphi\}$ according to the lexicographic ordering on the tuples $M_{1^\varphi \ldots (l-1)^\varphi, j}$ with $j \in \{l^\varphi, \ldots, n^\varphi\}$, or to put it differently, to construct $\mathsf{tu}_{l-1}^\varphi(M)$. During traversal, we need to compute $\mathsf{lex}_{l-1}^\varphi(M)$ for many values of l and φ, that is, for each right coset considered. Performing the actual sort at each step would clearly be to slow and therefore would undo the advantages of these modifications. Fortunately, the fact that M is known always to be lexically ordered, will allow us to do these computations in a more efficient manner.

Lemma 5.4.9. *Let $M \in \mathcal{M}_n$ be lexically ordered, $i \in \{1, \ldots, n-1\}$ such that $p, r, q \in \{i+1, \ldots, n\}$ and $p \le r \le q$. If $\mathsf{cell}(\mathsf{lex}_i(M), p) = \mathsf{cell}(\mathsf{lex}_i(M), q) = a$ then $\mathsf{cell}(\mathsf{lex}_i(M), r) = a$.*

Proof : As M is lexically ordered, we have $M_{1\ldots i,p} \le M_{1\ldots i,r} \le M_{1\ldots i,q}$, or equivalently $p \overset{i,M}{\lesssim} r \overset{i,M}{\lesssim} q$. Since $\mathsf{cell}(\mathsf{lex}_i(M), p) = \mathsf{cell}(\mathsf{lex}_i(M), q)$ we have $p \overset{i,M}{\sim} q$. Therefore $p \overset{i,M}{\sim} r \overset{i,M}{\sim} q$ and thus $\mathsf{cell}(\mathsf{lex}_i(M), r) = a$. ∎

In other words, any cell of a lexically ordered partition is a complete *interval*, uniquely determined by its minimal and maximal element.

Lemma 5.4.10. *Let $M \in \mathcal{M}_n$ be lexically ordered, $i \in \{1, \ldots, n-1\}$ such that $p, q \in \{i+1, \ldots, n\}$. Then $\mathsf{cell}(\mathsf{lex}_i(M), p) \le \mathsf{cell}(\mathsf{lex}_i(M), q)$ if and only if $p \le q$.*

Proof : If $p \leq q$ then $M_{1\ldots i,p} \leq M_{1\ldots i,q}$ as M is lexically ordered. Hence $p \overset{i,M}{\lesssim} q$ and therefore $\mathsf{cell}(\mathsf{lex}_i(M), p) \leq \mathsf{cell}(\mathsf{lex}_i(M), q)$. Conversely, if $\mathsf{cell}(\mathsf{lex}_i(M), p) \leq \mathsf{cell}(\mathsf{lex}_i(M), q)$ then $p \overset{i,M}{\lesssim} q$, or equivalently $M_{1\ldots i,p} \leq M_{1\ldots i,q}$. Since M is lexically ordered we find that $p \leq q$. ■

The fact that M is lexically ordered tells us something about the structure of $\mathsf{tu}_i(M)$. We have

$$\mathsf{tu}_i(M) = (M_{1\ldots i,i+1}, M_{1\ldots i,i+2}, \ldots, M_{1\ldots i,n})$$

We defer the low-level discussion of the algorithm and data structure which we use to determine lexically ordered image partitions during traversal, till the next section. In the remainder of this section, we will describe, at a higher conceptual level the actual modifications to the canonicity algorithm itself. In order to do so, we introduce three extra concepts. The *tentative error level* $\mathsf{err}_i^\pi(M)$ identifies the first tuple in which $\mathsf{tu}_i^\pi(M)$ and $\mathsf{tu}_i(M)$ differ, whereas the corresponding *tentative pivot level* $\mathsf{piv}_i^\pi(M)$ serves as identification of the first distinct entry of these tuples. Finally the tentative action $\mathsf{act}_i^\pi(M)$ identifies the order of these two entries.

Definition 5.4.11. Let $M \in \mathcal{M}_n$ be lexically ordered, $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \ldots, n-1\}$ such that $\mathsf{tu}_i^\pi(M) \neq \mathsf{tu}_i(M)$. Let c be the index of the first two tuples, $\mathsf{tu}_i^\pi(M)_c$ and $\mathsf{tu}_i(M)_c$, in which both sequences differ, then we define $\mathsf{err}_i^\pi(M) = i + c$. ◇

Definition 5.4.12. Let $M \in \mathcal{M}_n$ be lexically ordered, $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \ldots, n-1\}$ such that $\mathsf{tu}_i^\pi(M) \neq \mathsf{tu}_i(M)$. Let $\mathsf{err}_i^\pi(M) = i + c$, then we define $\mathsf{piv}_i^\pi(M) = q$ where $q \in \{1, \ldots, i\}$ such that $\mathsf{tu}_i^\pi(M)_c = (x_1, \ldots, x_i)$, $\mathsf{tu}_i(M)_c = (y_1, \ldots, y_i)$ satisfying $(x_1, \ldots, x_{q-1}) = (y_1, \ldots, y_{q-1})$ and $x_q \neq y_q$. ◇

Definition 5.4.13. Let $M \in \mathcal{M}_n$ be lexically ordered, $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \ldots, n-1\}$ such that $\mathsf{tu}_i^\pi(M) \neq \mathsf{tu}_i(M)$. Let $\mathsf{err}_i^\pi(M) = i + c$ and $\mathsf{piv}_i^\pi(M) = q \in \{1, \ldots, i\}$. Let $\mathsf{tu}_i^\pi(M)_c = (x_1, \ldots, x_i)$ and $\mathsf{tu}_i(M)_c = (y_1, \ldots, y_i)$. Then we define $\mathsf{act}_i^\pi(M) = 1$ if $x_q > y_q$ whereas $\mathsf{act}_i^\pi(M) = -1$ if $x_q < y_q$. ◇

As before we write $\mathsf{err}_i(M)$, $\mathsf{piv}_i(M)$ and $\mathsf{act}_i(M)$ instead of resp. $\mathsf{err}_i^{\mathrm{id}}(M)$, $\mathsf{piv}_i^{\mathrm{id}}(M)$ and $\mathsf{act}_i^{\mathrm{id}}(M)$. Again $\mathsf{err}_i^\pi(M)$, $\mathsf{piv}_i^\pi(M)$ and $\mathsf{piv}_i^\pi(M)$ only depend on the i-prefix of π. For future notational simplicity we set $\mathsf{err}_i^\pi(M) = n + 1$, $\mathsf{piv}_i^\pi(M) = 0$ and $\mathsf{act}_i^\pi(M) = 0$ when $\mathsf{tu}_i^\pi(M) = \mathsf{tu}_i(M)$. Clearly $\mathsf{err}_i(M) = n + 1$, $\mathsf{piv}_i(M) = 0$ and $\mathsf{act}_i(M) = 0$.

Example 5.17. Let π and M as in Example 5.11 and recall that

$$\begin{array}{rclcrcl} \mathsf{tu}_1(M) &=& ((1),(1),(1),(\mathbf{1})) & & \mathsf{tu}_2(M) &=& ((1,1),(1,\mathbf{1}),(1,2)) \\ \mathsf{tu}_1^\pi(M) &=& ((1),(1),(1),(\mathbf{2})) & & \mathsf{tu}_2^\pi(M) &=& ((1,1),(1,\mathbf{2}),(2,1)) \end{array}$$

The definitions imply that $\mathsf{err}_1^\pi(M) = 5$, $\mathsf{piv}_1^\pi(M) = 1$ and $\mathsf{act}_1^\pi(M) = 1$. Also $\mathsf{err}_2^\pi(M) = 4$, $\mathsf{piv}_2^\pi(M) = 2$ and $\mathsf{act}_2^\pi(M) = 1$. •

Example 5.18. Let π and M as in Example 5.12 and recall that

$$\begin{array}{rclcrcl} \mathsf{tu}_1(M) &=& ((1),(\mathbf{2}),(2),(2)) & & \mathsf{tu}_2(M) &=& ((\mathbf{2},2),(2,2),(2,2)) \\ \mathsf{tu}_1^\pi(M) &=& ((1),(\mathbf{1}),(2),(2)) & & \mathsf{tu}_2^\pi(M) &=& ((\mathbf{1},2),(2,2),(2,2)) \end{array}$$

The definitions imply that $\mathsf{err}_1^\pi(M) = 3$, $\mathsf{piv}_1^\pi(M) = 1$ and $\mathsf{act}_1^\pi(M) = -1$. Also $\mathsf{err}_2^\pi(M) = 3$, $\mathsf{piv}_2^\pi(M) = 1$ and $\mathsf{act}_2^\pi(M) = -1$. •

The following theorems make us of these concepts above to express the central idea behind the modifications to our canonicity algorithm.

Theorem 5.4.14. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1,\dots,n-1\}$ such that $(\pi M)_i = M_i$. If $\mathsf{err}_i^\pi(M) = i+1$ and $\mathsf{act}_i^\pi(M) = 1$, then*

$$(h\,M)_{i+1} > M_{i+1}$$

for all $h \in U^{(i)}\pi$. Moreover we have $\mathsf{err}^h(M) = \mathsf{err}_i^\pi(M)$ and $0 < \mathsf{piv}^h(M) \leq \mathsf{piv}_i^\pi(M)$ for each such h.

Proof : Write $l = \mathsf{piv}_i^\pi(M)$. Because $\mathsf{err}_i^\pi(M)$ and $\mathsf{piv}_i^\pi(M)$ only depend on the i-prefix of π, it is sufficient to prove the theorem for $h = \pi$. The special case $\mathsf{err}_i^\pi(M) = i+1$ and $\mathsf{act}_i^\pi(M) = 1$ means that $\mathsf{tu}_i^\pi(M)_{[1]}$ must be lexicographically larger than $\mathsf{tu}_i(M)_{[1]}$, which, because M is lexically ordered, is equal to $M_{1\dots i,i+1}$. Since $\mathsf{tu}_i(M)_{[1]} < \mathsf{tu}_i^\pi(M)_{[1]} \leq \mathsf{tu}_i^\pi(M)_{[a]}$ for each $a \in \{1,\dots,|\mathsf{lex}_i^\pi(M)|\}$ we have $M_{1^\pi\dots i^\pi,j} > M_{1\dots i,i+1}$ for all $j \in \{(i+1)^\pi,\dots,n^\pi\}$. Note that l indicates the first entry in which both tuples differ. More precisely for each $p \in \mathsf{lex}_i^\pi(M)_{[1]}$ we have $M_{1^\pi\dots l^\pi,p} > M_{1\dots l,i+1}$, hence also $M_{1^\pi\dots l^\pi,j} > M_{1\dots l,i+1}$. In particular this is true for $j = (i+1)^{\pi^{-1}}$, hence

$$(\pi M)_{1\dots l,i+1} > M_{1\dots l,i+1}.$$

As $(\pi M)_i = M_i$, it follows that $(\pi M)_{i+1} > M_{i+1}$. Moreover $\mathsf{err}^\pi(M) = \mathsf{err}_i^\pi(M)$ and $0 < \mathsf{piv}^\pi(M) \leq \mathsf{piv}_i^\pi(M)$. ■

Theorem 5.4.15. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \dots, n-1\}$ such that $(\pi M)_i = M_i$. If $\mathsf{err}_i^\pi(M) = i+1$ and $\mathsf{act}_i^\pi(M) = -1$, then*

$$(h\,M)_{i+1} < M_{i+1}$$

for all $h \in U^{(i)}\,\pi$ such that $(i+1)^h \in \mathsf{lex}_i^\pi(M)_{[1]}$. Moreover we have $\mathsf{err}^h(M) = \mathsf{err}_i^\pi(M)$ and $\mathsf{piv}^h(M) = \mathsf{piv}_i^\pi(M)$ for each such h.

Proof : Write $l = \mathsf{piv}_i^\pi(M)$. Note that $\mathsf{lex}_i^\pi(M)_{[1]}$ only depends on the i-prefix of π. Hence as in Theorem 5.4.14, it is sufficient to prove the theorem for $h = \pi$ with $(i+1)^\pi \in \mathsf{lex}_i^\pi(M)_{[1]}$. The special case $\mathsf{err}_i^\pi(M) = i+1$ and $\mathsf{act}_i^\pi(M) = -1$ means that $\mathsf{tu}_i^\pi(M)_{[1]}$ must be lexicographically smaller than $\mathsf{tu}_i(M)_{[1]}$, which, because M is lexically ordered, is equal to $M_{1\dots l,i+1}$. Note that l indicates the first entry in which both tuples differ. More precisely for each $p \in \mathsf{lex}_i^\pi(M)_{[1]}$ we have $M_{1^\pi\dots(l-1)^\pi,p} = M_{1,\dots l-1,i+1}$ and $M_{l^\pi,p} < M_{l,i+1}$. In particular this is true for $p = (i+1)^{\pi^{-1}}$, hence

$$(\pi\,M)_{1,\dots l-1,i+1} = M_{1,\dots l-1,i+1} \quad \text{and} \quad (\pi\,M)_{l,i+1} < M_{l,i+1}.$$

As $(\pi\,M)_i = M_i$, it follows that $(\pi\,M)_{i+1} < M_{i+1}$. Moreover $\mathsf{err}^\pi(M) = \mathsf{err}_i^\pi(M)$ and $\mathsf{piv}^\pi(M) = \mathsf{piv}_i^\pi(M)$ ■

Theorem 5.4.16. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \dots, n-1\}$ such that $(\pi M)_i = M_i$. If $\mathsf{err}_i^\pi(M) > i+1$, then for all $h \in U^{(i)}\,\pi$ we have*

$$(h\,M)_{i+1} = M_{i+1}$$

when $(i+1)^h \in \mathsf{lex}_i^\pi(M)_{[1]}$. Otherwise when $(i+1)^h \notin \mathsf{lex}_i^\pi(M)_{[1]}$ we have

$$(h\,M)_{i+1} > M_{i+1}.$$

Moreover $\mathsf{err}^h(M) = i+1$ and $0 < \mathsf{piv}^h(M) \le i$ for each such h with $(i+1)^h \notin \mathsf{lex}_i^\pi(M)_{[1]}$.

Proof : As in Theorem 5.4.15, it is sufficient to prove the theorem for $h = \pi$. The special case $\mathsf{err}_i^\pi(M) > i+1$ means that $\mathsf{tu}_i^\pi(M)_{[1]}$ must be equal to $\mathsf{tu}_i(M)_{[1]}$, which, because M is lexically ordered, is equal to $M_{1\dots l,i+1}$. More precisely for each $p \in \mathsf{lex}_i^\pi(M)_{[1]}$ we have $M_{1^\pi \dots i^\pi, p} = M_{1\dots i, i+1}$. If $(i+1)^\pi \in \mathsf{lex}_i^\pi(M)_{[1]}$, then in particular this is true for $p = (i+1)^{\pi^{-1}}$, hence

$$(\pi M)_{1\dots i,i+1} = M_{1\dots i,i+1}.$$

As $(\pi M)_i = M_i$, it follows that $(\pi M)_{i+1} = M_{i+1}$. Moreover we have $\mathsf{tu}_i(M)_{[1]} = \mathsf{tu}_i^\pi(M)_{[1]} < \mathsf{tu}_i^\pi(M)_{[b]}$ for each $b \in \{2, \dots, |\mathsf{lex}_i^\pi(M)|\}$. Hence we find that $M_{1^\pi \dots i^\pi, r} > M_{1\dots i,i+1}$ for each $r \in \{(i+1)^\pi, \dots, n^\pi\} \setminus \mathsf{lex}_i^\pi(M)_{[1]}$. If $(i+1)^\pi \notin \mathsf{lex}_i^\pi(M)_{[1]}$, then in particular this is true for $r = (i+1)^{\pi^{-1}}$, hence

$$(\pi M)_{1\dots i,i+1} > M_{1\dots i,i+1}.$$

As $(\pi M)_i = M_i$, it follows that $(\pi M)_{i+1} > M_{i+1}$. Moreover $\mathsf{err}^h(M) = i+1$ and $0 < \mathsf{piv}^h(M) \leq i$ for each such h with $(i+1)^h \notin \mathsf{lex}_i^\pi(M)_{[1]}$. ■

The concepts of tentative error and pivot level and tentative action allow us to extend the central idea behind the modifications still somewhat further. Theorems 5.4.14 and 5.4.15 both assume that $\mathsf{err}_i^\pi(M) = i+1$. Similar theorems can be established under the assumption that $\mathsf{err}_i^\pi(M) = i+2$. In order to do so, we first need to introduce the following theorem.

Theorem 5.4.17. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\varphi, \rho \in \mathrm{Sym}(n)$ and $l \in \{2, \dots, n-1\}$ such that $\varphi \in U^{(l-1)}\rho$ with $l^\varphi \in \mathsf{lex}_{l-1}^\rho(M)_{[1]}$. If $\mathsf{err}_{l-1}^\rho(M) > l$ then $\mathsf{err}_l^\varphi(M) \leq \mathsf{err}_{l-1}^\rho(M)$. Moreover, $\mathsf{err}_l^\varphi(M) = \mathsf{err}_{l-1}^\rho(M)$ if and only if $\mathsf{piv}_l^\varphi(M) = \mathsf{piv}_{l-1}^\rho(M)$, while $\mathsf{err}_l^\varphi(M) < \mathsf{err}_{l-1}^\rho(M)$ if and only if $\mathsf{piv}_l^\varphi(M) = l$.*

Proof : As $\mathsf{err}_{l-1}^\rho(M)$ and $\mathsf{lex}_{l-1}^\rho(M)$ only depend on the $(l-1)$-prefix of ρ and $\varphi \in U^{(l-1)}\rho$, it is equivalent to prove the theorem for $\rho = \varphi$. Note that as M is lexically ordered, the a-th tuple of $\mathsf{tu}_l(M)$ is equal to $M_{1\dots l,l+a}$. There exists a set $\{a_{l+1}, \dots, a_n\} = \{(l+1)^\varphi, \dots, n^\varphi\}$ such that $\mathsf{tu}_l^\varphi(M)_k = M_{1^\varphi \dots l^\varphi, a_k}$ for each $k \in l+1, \dots, n$. Write $e = \mathsf{err}_l^\varphi(M)$. This means that $\mathsf{tu}_l^\varphi(M)_{j-l} = \mathsf{tu}_l(M)_{j-l}$ for each $j \in \{l+1, \dots, e-l-1\}$. More precisely, $M_{1^\varphi \dots l^\varphi, a_j} = M_{1\dots l,j}$ for each such j. Moreover, $M_{1^\varphi \dots l^\varphi, a_j} \leq M_{1^\varphi \dots l^\varphi, a_p}$ for each $p \in \{e, \dots, n\}$. Hence for each such j and p we have

$$M_{1^\varphi \dots (l-1)^\varphi, a_j} = M_{1\dots l-1,j} \quad \text{and} \quad M_{1^\varphi \dots (l-1)^\varphi, a_j} \leq M_{1^\varphi \dots (l-1)^\varphi, a_p} \tag{5.17}$$

Besides, since $\mathsf{err}_{l-1}^\varphi(M) > l$ and $l^\varphi \in \mathsf{lex}_{l-1}^\varphi(M)_{[1]}$ we have

$$(\varphi M)_{1\dots l-1,l} = M_{1\dots l-1,l} \quad \text{and} \quad (\varphi M)_{1\dots l-1,l} \leq M_{1^\varphi \dots (l-1)^\varphi, a_q} \tag{5.18}$$

for all $q \in \{l+1, \dots, n\}$. Combining (5.17) and (5.18) we find that at least the first $e-l$ tuples of $\mathsf{tu}^{\varphi}_{l-1}(M)$ and $\mathsf{tu}_{l-1}(M)$ must coincide. Hence $\mathsf{err}^{\varphi}_{l}(M) \leq \mathsf{err}^{\varphi}_{l-1}(M)$.

If $\mathsf{err}^{\varphi}_{l}(M) = n+1$ then clearly $\mathsf{err}^{\varphi}_{l-1}(M) = n+1$. Hence we have $\mathsf{piv}^{\varphi}_{l}(M) = \mathsf{piv}^{\varphi}_{l-1}(M) = 0$. Otherwise, if $\mathsf{err}^{\varphi}_{l}(M) < n+1$, then $\mathsf{tu}^{\varphi}_{l}(M)_{e-l} \neq \mathsf{tu}_{l}(M)_{e-l}$. More precisely, we have

$$M_{1^{\varphi}\dots l^{\varphi},a_e} \neq M_{1\dots l,e} \quad \text{and} \quad M_{1^{\varphi}\dots l^{\varphi},a_e} \leq M_{1^{\varphi}\dots l^{\varphi},a_s}. \tag{5.19}$$

for each $s \in \{e+1, \dots, n\}$. We distinguish between the following cases:

- If $M_{1\dots l-1,e} = M_{1^{\varphi}\dots(l-1)^{\varphi},a_e}$, then $M_{l,e} \neq M_{l^{\varphi},s}$. Hence $\mathsf{piv}^{\varphi}_{l}(M) = l$. Moreover, as $M_{1^{\varphi}\dots(l-1)^{\varphi},a_e} \leq M_{1^{\varphi}\dots(l-1)^{\varphi},a_s}$, we find that $M_{1^{\varphi}\dots(l-1)^{\varphi},a_e} = \mathsf{tu}^{\varphi}_{l-1}(M)_{e-l+1}$, combining (5.17) and (5.18). Hence $\mathsf{err}^{\varphi}_{l}(M) < \mathsf{err}^{\varphi}_{l-1}(M)$.
- If $M_{1\dots l-1,e} \neq M_{1^{\varphi}\dots(l-1)^{\varphi},s}$, then clearly $1 \leq \mathsf{piv}^{\varphi}_{l}(M) < l$. Moreover, as $M_{1^{\varphi}\dots(l-1)^{\varphi},a_e} \leq M_{1^{\varphi}\dots(l-1)^{\varphi},a_s}$, we find that
$$M_{1^{\varphi}\dots(l-1)^{\varphi},a_e} = \mathsf{tu}^{\varphi}_{l-1}(M)_{e-l+1},$$
combining (5.17) and (5.18). Hence we have $\mathsf{err}^{\varphi}_{l}(M) = \mathsf{err}^{\varphi}_{l-1}(M)$ and $\mathsf{piv}^{\varphi}_{l-1}(M) = \mathsf{piv}^{\varphi}_{l}(M)$.

■

Corollary 5.4.1. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\varphi, \rho \in \mathrm{Sym}(n)$ and $l \in \{2, \dots, n-1\}$ such that $\varphi \in U^{(l-1)}\rho$ with $l^{\varphi} \in \mathsf{lex}^{\rho}_{l-1}(M)_{[1]}$ and such that $\mathsf{err}^{\rho}_{l-1}(M) > l$. Write $e = \mathsf{err}^{\varphi}_{l}(M)$. If $\mathsf{piv}^{\varphi}_{l}(M) = \mathsf{piv}^{\rho}_{l-1}(M)$ then $\mathsf{act}^{\varphi}_{l}(M) = \mathsf{act}^{\rho}_{l-1}(M)$. Otherwise, if $\mathsf{piv}^{\varphi}_{l}(M) = l$ then $\mathsf{act}^{\varphi}_{l}(M) = 1$ if $x_l > y_l$ whereas $\mathsf{act}^{\varphi}_{l}(M) = -1$ if $x_l < y_l$ where $\mathsf{tu}^{\varphi}_{l}(M)_{e-l} = (x_1, \dots, x_l)$ and $\mathsf{tu}_{i}(M)_{e-l} = (y_1, \dots, y_l)$.* ◇

The following theorem extends the central idea behind Theorem 5.4.14.

Theorem 5.4.18. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \dots, n-2\}$ such that $(\pi M)_i = M_i$. If $\mathsf{err}^{\pi}_{i}(M) = i+2$ and $\mathsf{act}^{\pi}_{i}(M) = 1$, then*

$$(h\,M)_{i+2} > M_{i+2}$$

for all $h \in U^{(i)}\pi$. Moreover we have $\mathsf{err}^{h}(M) \leq \mathsf{err}^{\pi}_{i}(M)$ and $0 < \mathsf{piv}^{h}(M) \leq \mathsf{piv}^{\pi}_{i}(M)$ for each such h.

Proof : As in Theorem 5.4.15, it is sufficient to prove the theorem for $h = \pi$. Let $(i+1)^\pi \notin \mathsf{lex}_i^\pi(M)_{[1]}$. Since $(\pi M)_i = M_i$ and $\mathsf{err}_i^\pi(M) > i+1$, Theorem 5.4.16 states that

$$(\pi M)_{i+1} > M_{i+1}, \quad \mathsf{err}^\pi(M) = i+1 \quad \text{and} \quad 0 < \mathsf{piv}^i(M) \le i.$$

We still have to prove that $\mathsf{piv}^\pi(M) \le \mathsf{piv}_i^\pi(M)$. Write $l = \mathsf{piv}_i^\pi(M)$. Then $\mathsf{tu}_i^\pi(M)_1 = \mathsf{tu}_i(M)_1$ while $\mathsf{tu}_i^\pi(M)_2 > \mathsf{tu}_i(M)_2$. Because M is lexically ordered, $\mathsf{tu}_i(M)_1$ and $\mathsf{tu}_i(M)_2$ are resp. equal to $M_{1\ldots l,i+1}$ and $M_{1\ldots l,i+2}$. Clearly $\mathsf{tu}_i^\pi(M)_1 = \mathsf{tu}_i^\pi(M)_{[1]}$ and $\mathsf{tu}_i^\pi(M)_2 = \mathsf{tu}_i^\pi(M)_{[2]}$. Hence $\mathsf{tu}_i(M)_2 < \mathsf{tu}_i^\pi(M)_{[a]}$ for each $a \in \{2, \ldots, |\mathsf{lex}_i^\pi(M)|\}$. More precisely, we have $M_{1\ldots l,i+2} < M_{1^\pi \ldots l^\pi,r}$ for all $r \in \{(i+1)^\pi, \ldots, n^\pi\} \setminus \mathsf{lex}_i^\pi(M)_{[1]}$. Since M is lexically ordered also $M_{1\ldots l,i+1} < M_{1^\pi \ldots l^\pi,r}$ for each such r. In particular this is true for $r = (i+1)^{\pi^{-1}}$, hence

$$M_{1\ldots l,i+1} < (\pi M)_{1\ldots l,i+1}.$$

As $(\pi M)_i = M_i$, it follows readily that $\mathsf{piv}^\pi(M) \le \mathsf{piv}_i^\pi(M)$.

Otherwise, let $(i+1)^\pi \in \mathsf{lex}_i^\pi(M)_{[1]}$. Since $(\pi M)_i = M_i$ and $\mathsf{err}_i^\pi(M) > i+1$, Theorem 5.4.16 states that $(\pi M)_{i+1} = M_{i+1}$. Moreover, according to Theorem 5.4.17 we have

$$\mathsf{err}_{i+1}^\pi(M) = \mathsf{err}_i^\pi(M) \quad \text{and} \quad \mathsf{piv}_{i+1}^\pi(M) = \mathsf{piv}_i^\pi(M).$$

Hence substituting $i+1$ for i in Theorem 5.4.14 we find that $(\pi M)_{i+2} > M_{i+2}$, $\mathsf{err}^\pi(M) = i+2$ and $\mathsf{piv}^\pi(M) \le \mathsf{piv}_{i+1}^\pi(M) = \mathsf{piv}_i^\pi(M)$. ■

The following theorem extends the central idea behind Theorem 5.4.15.

Theorem 5.4.19. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \ldots, n-2\}$ such that $(\pi M)_i = M_i$. Let $\tau \in U^{(i)}\pi$ such that $(i+1)^\tau \in \mathsf{lex}_i^\pi(M)_{[1]}$. If $\mathsf{err}_i^\pi(M) = i+2$ and $\mathsf{act}_i^\pi(M) = -1$, then*

$$(h M)_{i+2} < M_{i+2}$$

for all $h \in U^{(i+1)}\tau$ such that $(i+2)^h \in \mathsf{lex}_{i+1}^\tau(M)_{[1]}$. Moreover $\mathsf{err}^h(M) = \mathsf{err}_i^\pi(M)$ and $\mathsf{piv}^h(M) = \mathsf{piv}_i^\pi(M)$ for each such h.

Proof : Note that $\mathsf{lex}_{i+1}^\tau(M)_{[1]}$ only depends on the $(i+1)$-prefix of τ. Hence it is sufficient to proof the theorem for $h = \tau$ with $(i+2)^\tau \in \mathsf{lex}_{i+1}^\tau(M)_{[1]}$.

Moreover, as $\tau \in U^{(i)}\,\pi$ and $\mathsf{lex}_i^{\pi}(M)_{[1]}$ only depends on the i-prefix of π, it is sufficient to proof the theorem for $h = \pi$ with $(i+1)^{\pi} \in \mathsf{lex}_i^{\pi}(M)_{[1]}$ and $(i+2)^{\pi} \in \mathsf{lex}_{i+1}^{\pi}(M)_{[1]}$. Since $\pi \in \mathsf{lex}_i^{\pi}(M)_{[1]}$, $(\pi\, M)_i = M_i$ and $\mathsf{err}_i^{\pi}(M) > i+1$ we find according to Theorem 5.4.16 that $(\pi\, M)_{i+1} = M_{i+1}$. Moreover, according to Theorem 5.4.17 we have

$$\mathsf{err}_{i+1}^{\pi}(M) = \mathsf{err}_i^{\pi}(M) \quad \text{and} \quad \mathsf{piv}_{i+1}^{\pi}(M) = \mathsf{piv}_i^{\pi}(M).$$

Hence, as $(i+2)^{\pi} \in \mathsf{lex}_{i+1}^{\pi}(M)_{[1]}$ we that $(\pi\, M)_{i+2} < M_{i+2}$, $\mathsf{err}^{\pi}(M) = i+2$ and $\mathsf{piv}^{\pi}(M) = \mathsf{piv}_{i+1}^{\pi}(M) = \mathsf{piv}_i^{\pi}(M)$, by Theorem 5.4.15. ■

Each of the theorems introduced above shall play an important role in the improvements Algorithm 5.5 makes on Algorithm 5.4. The differences between these canonicity algorithms are essentially the following:

- The function **diffStab(*i*)** checks whether a permutation $g \in U^{(i-1)} - U^{(i)}$ can be found such that $g\, M < M$. In order to do so, all right cosets in

$$U^{(i)}\,(i\;i+1),\; U^{(i)}\,(i\;i+2), \ldots,\; U^{(i)}\,(i\;n-1),\; U^{(i)}\,(i\;n) \qquad (5.20)$$

  are successively considered. Since $(g\; M)_{i-1} = M_{i-1}$ and $\mathsf{err}_{i-1}(M) = n+1$, we can easily restrict the number of right cosets in (5.20) we have to consider. On the one hand Theorem 5.4.16 states that

$$(g\, M)_i > M_i$$

  for all $g \in U^{(i)}\,(i\;k)$ with $k \notin \mathsf{lex}_{i-1}(M)_{[1]}$. Hence no such g could ever lead to a counterexample to the minimality of M. Therefore each such right coset $U^{(i)}\,(i\,k)$ can be discarded from traversal. Note that $\mathsf{err}^{g}(M) = i$ and $\mathsf{piv}^{g}(M) \leq i-1$ for each such g. On the other hand Theorem 5.4.16 states that

$$(g\, M)_i = M_i$$

  for all $g \in U^{(i)}\,(i\;j)$ with $j \in \mathsf{lex}_{i-1}(M)_{[1]} \setminus \{i\}$. Hence each such g could still lead to a counterexample to the minimality of M.

  The canonicity algorithm proceeds by successively considering each of these right cosets $U^{(i)}\,(i\;j)$. Before we traverse $U^{(i)}\,(i\;j)$, we first invoke the method **refine(**$(i\;j)$, i**)** (❷ in Algorithm 5.5). By calling this method, $\mathsf{lex}_i^{(i\,j)}(M)$, $\mathsf{err}_i^{(i\,j)}(M)$, $\mathsf{piv}_i^{(i\,j)}(M)$ and $\mathsf{act}_i^{(i\,j)}(M)$ are computed. For a detailed description of these computations we refer to the next section. Finally, note that these right cosets $U^{(i)}\,(i\;j)$ still need to be traversed in increasing order of j; otherwise the “minimal in orbit” criterion could no longer be applied.

- The function **rightCoset(π, i, s)** checks whether any permutation $g \in U^{(i-1)}\,\pi$ can be found such that $g\,M < M$. If in Algorithm 5.4 the function **rightCoset(π, i, s)** is invoked with $i = n+1$ then $\pi \in (\operatorname{Aut} M)^{(s)}$. Theorem 5.4.16 allows us to alter the way in which we detect such automorphisms. More precisely, if $i = n$ and $\mathsf{err}^{\pi}_{i-1}(M) > n$ (❽ in Algorithm 5.5), then we find according to Theorem 5.4.16 that $(\pi\,M)_n = M_n$. Hence $\pi \in (\operatorname{Aut} M)^{(s)}$. We return $\mathsf{act}^{\pi}_{i-1}(M) = 0$ so that the canonicity algorithm proceeds correctly (❸ in Algorithm 5.5).

  Otherwise, if in Algorithm 5.4 the function **rightCoset(π, i, s)** is invoked with $i \neq n+1$, then the function successively considers all right cosets in

$$U^{(i)}\,\pi,\ U^{(i)}\,(i\;i+1)\,\pi, \ldots, U^{(i)}\,(i\;n-1)\,\pi,\ U^{(i)}\,(i\;n)\,\pi. \tag{5.21}$$

  Theorems 5.4.14–5.4.19 allow us to restrict the number of right cosets in (5.21) we have to consider. We distinguish between the following cases:

  1. If $\mathsf{err}^{\pi}_{i-1}(M) = i$ (❼ in Algorithm 5.5) and $\mathsf{act}^{\pi}_{i-1}(M) = -1$, then Theorem 5.4.15 states that

$$(g\,M)_i < M_i$$

     for all $g \in U^{(i)}\,(i\;k^{\pi^{-1}})\,\pi$ with $k \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$. Hence each such g is a counterexample to the minimality of M. We return $\mathsf{act}^{\pi}_{i-1}(M) = -1$ so that the canonicity algorithm terminates correctly (❸ in Algorithm 5.5). Note that $\mathsf{err}^{g}(M) = i$ and $\mathsf{piv}^{g}(M) = \mathsf{piv}^{\pi}_{i-1}(M)$ for each such g.

  2. Let $\tau = (i\;q^{\pi^{-1}})\,\pi$ with $q \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$. If $\mathsf{err}^{\pi}_{i-1}(M) = i+1$ (❼ in Algorithm 5.5) and $\mathsf{act}^{\pi}_{i-1}(M) = -1$, then Theorem 5.4.19 states that

$$(g\,M)_{i+1} < M_{i+1}$$

     for all $g \in U^{(i+1)}\,(i+1\;p^{\tau^{-1}})\,\tau$ with $p \in \mathsf{lex}^{\tau}_{i}(M)_{[1]}$. Hence each such g is a counterexample to the minimality of M. We return $\mathsf{act}^{\pi}_{i-1}(M) = -1$ so that the canonicity algorithm terminates correctly (❸ in Algorithm 5.5). Note that $\mathsf{err}^{g}(M) = i+1$ and $\mathsf{piv}^{g}(M) = \mathsf{piv}^{\pi}_{i-1}(M)$ for each such g.

3. If $\mathsf{err}^{\pi}_{i-1}(M) = i$ (❼ in Algorithm 5.5) and $\mathsf{act}^{\pi}_{i-1}(M) = 1$, then Theorem 5.4.14 states that

$$(g\,M)_i > M_i$$

for all $g \in U^{(i-1)}\,\pi$. Hence $U^{(i-1)}\,\pi$ does not contain a counterexample to the minimality of M. Therefore each right coset in (5.21) can be discarded. We return $\mathsf{act}^{\pi}_{i-1}(M) = 1$ so that the canonicity algorithm proceeds correctly (❸ in Algorithm 5.5). Note that $\mathsf{err}^{g}(M) = i$ and $0 < \mathsf{piv}^{g}(M) \leq \mathsf{piv}^{\pi}_{i-1}(M)$ for each such g.
4. If $\mathsf{err}^{\pi}_{i-1}(M) = i+1$ (❼ in Algorithm 5.5) and $\mathsf{act}^{\pi}_{i-1}(M) = 1$, then Theorem 5.4.18 states that

$$(g\,M)_{i+1} > M_{i+1}$$

for all $g \in U^{(i-1)}\,\pi$. Hence no such g could ever lead to a counterexample to the minimality of M. Therefore each right coset in (5.21) can be discarded. We return $\mathsf{act}^{\pi}_{i-1}(M) = 1$ so that the canonicity algorithm proceeds correctly (❸ in Algorithm 5.5). Note that $\mathsf{err}^{g}(M) = i+1$ and $0 < \mathsf{piv}^{g}(M) \leq \mathsf{piv}^{\pi}_{i-1}(M)$ for each such g.
5. If $\mathsf{err}^{\pi}_{i-1}(M) > i+1$ (❹ in Algorithm 5.5) then on the one hand Theorem 5.4.16 states that

$$(g\,M)_i > M_i$$

for all $g \in U^{(i)}\,(i\;k^{\pi^{-1}})\,\pi$ with $k \in \{i^{\pi}, \ldots, n^{\pi}\} \setminus \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$. Hence no such g could ever lead to a counterexample to the minimality of M. Therefore each such right coset $U^{(i)}\,(i\;k^{\pi^{-1}})\,\pi$ can be discarded from traversal. Note that $\mathsf{err}^{g}(M) = i$ and $\mathsf{piv}^{g}(M) \leq i-1$ for each such g.

   On the other hand Theorem 5.4.16 states that

$$(g\,M)_i = M_i$$

for all $g \in U^{(i)}\,\pi'$ with $\pi = (i\;l^{\pi^{-1}})\,\pi$ and $l \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$. Hence each such g could still lead to a counterexample to the minimality of M. The canonicity algorithm proceeds by considering each of these right cosets $U^{(i)}\,\pi'$ (❺ in Algorithm 5.5). Before we traverse $U^{(i)}\,\pi'$, we first invoke the method **refine(π', i)** (❻ Algorithm 5.5). By calling this method, $\mathsf{lex}_i^{\pi'}(M)$, $\mathsf{err}_i^{\pi'}(M)$, $\mathsf{piv}_i^{\pi'}(M)$ and $\mathsf{act}_i^{\pi'}(M)$ are computed. We postpone a detailed, low-level description of these computations until the next section.

Write $e = \mathsf{err}_{i-1}^{\pi}(M)$. Theorem 5.4.17 states that $\mathsf{err}_i^{\pi'}(M) \leq e$ for each such π'. Hence by the recursive structure of the canonicity algorithm, we can predict even before we actually traverse the right coset $U^{(i-1)}\pi$ that we will never consider a right coset $U^{(k)}\tau \subset U^{(i-1)}\pi$ in the recursion at level $k > e-2$. Moreover, if we would encounter at level $e-2$ in the recursion a right coset $U^{(e-2)}\sigma \subset U^{(i-1)}\pi$, then $\mathsf{err}_{e-2}^{\sigma}(M) = e$.

In this particular case, Theorem 5.4.17 and Corollary 5.4.1 state that $\mathsf{piv}_{e-2}^{\sigma}(M) = \mathsf{piv}_{i-1}^{\pi}(M)$ and $\mathsf{act}_{e-2}^{\sigma}(M) = \mathsf{act}_{i-1}^{\pi}(M)$. Hence the action to be taken, that is, either discard $U^{(e-2)}\sigma$ entirely, terminate the canonicity test or detect that $\sigma \in (\operatorname{Aut} M)^{(s)}$, depends only on the value of $\mathsf{act}_{i-1}^{\pi}(M)$.

Finally the question remains whether we still obtain, as in Algorithm 5.4, a strong generating set $S = S^{(0)}$ with base $[1, \ldots, n]$ for $\operatorname{Aut} M$ when M is minimal. Assume that $\pi \in \operatorname{Aut} M$. Then for each $i \in \{1, \ldots, n-1\}$ we have $\mathsf{err}_i^{\pi}(M) = n+1$ because $M_{1\ldots i,p} = (\pi M)_{1\ldots i,p}$ for each $p \in \{i+1, \ldots, n\}$. Therefore the modifications introduced in Algorithm 5.5 never discard a right coset that contains an automorphism of M. Hence Algorithm 5.5 still obtains such a strong generating set S.

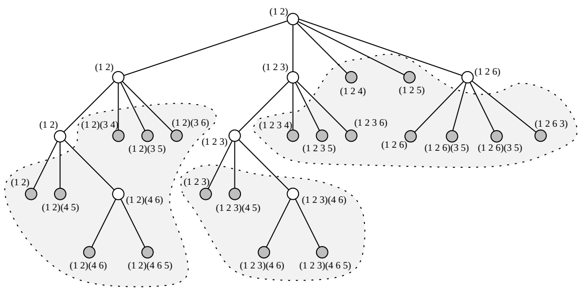
Below we present some examples to illustrate the points made in the previous discussion. We reconsider Examples 5.7, 5.9 and 5.10. These example serve to demonstrate that the improvements to our canonicity algorithm indeed avoid the drawbacks established in the previous section.

Example 5.19. Consider the lexically ordered matrix $M \in \mathcal{M}_6$ as in Example 5.7. Let $\sigma = (1\ 2)$ and $\pi = (1\ 2\ 6)$. The recursion tree shown corresponds to the traversal of $U^{(1)}\sigma = U^{(1)}\pi$ by Algorithm 5.4.

Algorithm 5.5 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function isCanonical($M \in \mathcal{M}_n$) : boolean
for $i \leftarrow n-1, \ldots 1$ do
    if diffStab($i$) $< 0$ then
        return false
return true

function diffStab($i$ : int) : int
for all $j \in (\mathsf{lex}_{i-1}(M)_{[1]} \setminus \{i\})$ do                    ❶
    if $j = \min(j^{\langle S^{(i-1)}_{\mathrm{Aut}M} \rangle})$ then
        refine($(i\ j)$, $i$)                                              ❷
        $d \leftarrow$ rightCoset($(i\ j)$, $i+1$, $i-1$)
        if $d < 0$ then
            return $d$
return 1

function rightCoset($\pi \in U^{(s)} - U^{(s+1)}, i, s$ : int) : int
if checkErrorLevel($\pi$, $i$, $s$) then
    return $\mathsf{act}^{\pi}_{i-1}(M)$                                    ❸
else                                                                       ❹
    for all $j \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$ do                     ❺
        $\pi' \leftarrow (i\ j^{\pi^{-1}})\ \pi$
        refine($\pi'$, $i$)                                                ❻
        $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
        if $d < 0 \lor d = 0$ then
            return $d$
    return 1

function checkErrorLevel($\pi \in U^{(s)} - U^{(s+1)}, i, s$ : int) : boolean
if $\mathsf{err}^{\pi}_{i-1}(M) \leq i+1$ then                               ❼
    if $\mathsf{err}^{\pi}_{i-1}(M) = n+1$ then                              ❽
        $\bar{S}^{(s)}_{\mathrm{Aut}\,M} \leftarrow \bar{S}^{(s)}_{\mathrm{Aut}\,M} \cup \pi$
    return true
else
    return false
```

$$M = \left(\begin{array}{cccccc} 0\,| & 1 & 1 & 1 & 1\,| & 2 \\ 1 & 0\,| & 1\,| & 2 & 2\,| & 1 \\ 1 & 1 & 0\,| & 2 & 2\,| & 2 \\ 1 & 2 & 2 & 0\,| & 1\,| & 1 \\ 1 & 2 & 2 & 1 & 0\,| & 2 \\ 2 & 1 & 2 & 1 & 2 & 0 \end{array}\right)$$



The shaded areas contain the right cosets which are now also discarded Algorithm 5.5.

1. We have $\mathsf{lex}_1^{\sigma}(M) = (\{1,3,6\},\{4,5\})$, $\mathsf{tu}_1(M) = \{(1),(1),(1),(\mathbf{1}),(2)\}$ and $\mathsf{tu}_1^{\sigma}(M) = \{(1),(1),(1),(\mathbf{2}),(2)\}$. Hence $\mathsf{err}_1^{\sigma}(M) = 5$, $\mathsf{piv}_1^{\sigma}(M) = 1$ and $\mathsf{act}_1^{\sigma}(M) = 1$. Therefore we find that only the right cosets $U^{(2)}\,(1\,2)$, $U^{(2)}\,(1\,2\,3)$ and $U^{(2)}\,(1\ 2\ 6)$ need to be considered because $\mathsf{lex}_1^{\sigma}(M)_{[1]} = \{1,3,6\}$.

2. Clearly $\mathsf{lex}_2^{\sigma}(M) = (\{3\},\{6\},\{4,5\})$, $\mathsf{tu}_2(M) = \{(1,1),(1,2),(\mathbf{1},2),(2,1)\}$ and $\mathsf{tu}_2^{\sigma}(M) = \{(1,1),(1,2),(\mathbf{2},1),(2,1)\}$. Hence $\mathsf{err}_2^{\sigma}(M) = 5$, $\mathsf{piv}_2^{\sigma}(M) = 1$, $\mathsf{act}_2^{\sigma}(M) = 1$. Therefore we find that only the right coset $U^{(3)}\,(1\,2)$ needs to be considered because $\mathsf{lex}_2^{\sigma}(M)_{[1]} = \{3\}$.

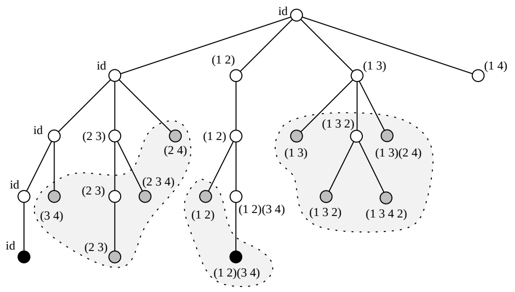
3. We have $\mathsf{lex}_3^{\sigma}(M) = (\{6\},\{4,5\})$, $\mathsf{tu}_3(M) = \{(1,2,2),(\mathbf{1},2,2),(2,1,2)\}$ and $\mathsf{tu}_3^{\sigma}(M) = \{(1,2,2),(\mathbf{2},1,2),(2,1,2)\}$. Hence we have $\mathsf{err}_3^{\sigma}(M) = 5$, $\mathsf{piv}_3^{\sigma}(M) = 1$ and $\mathsf{act}_3^{\sigma}(M) = 1$. Therefore we find that $(g\,M)_5 > M_5$ for all $g \in U^{(3)}\,\sigma$. Hence no such g could ever lead to a counterexample to the minimality of M.

4. Clearly $\mathsf{lex}_2^{\pi}(M) = (\{1,3\},\{4\},\{5\})$, $\mathsf{tu}_2(M) = \{(1,\mathbf{1}),(1,2),(1,2),(2,1)\}$, $\mathsf{tu}_2^{\pi}(M) = \{(1,\mathbf{2}),(1,2),(2,1),(2,2)\}$. Hence $\mathsf{err}_2^{\pi}(M) = 3$, $\mathsf{piv}_2^{\pi}(M) = 2$, $\mathsf{act}_2^{\pi}(M) = 1$. Therefore we find that $(g\,M)_3 > M_3$ for all $g \in U^{(2)}\,\pi$. Hence no such g could ever lead to a counterexample to the minimality of M.

•

Example 5.20. Consider the lexically ordered matrix $M \in \mathcal{M}_4$ below. The recursion tree shown corresponds to the traversal of Sym(4) by Algorithm 5.4.

$$M = \left(\begin{array}{cccc} 0\,\vert & 1 & 1\,\vert & 2 \\ 1 & 0\,\vert & 2\,\vert & 1 \\ 1 & 2 & 0\,\vert & 2 \\ 2 & 1 & 2 & 0 \end{array}\right)$$



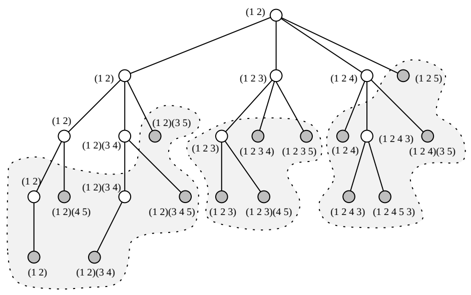
The shaded areas contain the right cosets which are now also discarded by Algorithm 5.5. Write $\pi = (1\ 3)$, $\sigma = (1\ 2\ 3)$ and $\sigma = (1\ 2)\ (3\ 4)$.

1. We have $\mathsf{lex}_1^{\pi}(M) = (\{1\}, \{2, 4\})$, $\mathsf{tu}_1(M) = \{(1), (\mathbf{1}), (1)\}$ and $\mathsf{tu}_1^{\sigma}(M) = \{(1), (\mathbf{2}), (2)\}$. Hence $\mathsf{err}_1^{\pi}(M) = 3$, $\mathsf{piv}_1^{\pi}(M) = 1$ and $\mathsf{act}_1^{\pi}(M) = 1$. Therefore we find that $(g\,M)_3 > M_3$ for each $g \in U^{(1)}\,\pi$. Hence no such g could ever lead to a counterexample of the minimality of M.
2. We have $\mathsf{lex}_1(M) = (\{2, 3\}, \{4\})$. Clearly $\mathsf{err}_1(M) = 5$, $\mathsf{piv}_1(M) = 0$ and $\mathsf{act}_1(M) = 0$. Hence we find that only the right coset $U^{(2)}\,(2\,3)$ needs to be traversed because $\mathsf{lex}_1(M)_{[1]} \setminus \{2\} = \{3\}$.
3. We have $\mathsf{lex}_3^{\sigma}(M) = (\{3\})$ and $\mathsf{tu}_3(M) = \mathsf{tu}_3^{\sigma}(M) = \{(2, 1, 2)\}$. Hence $\mathsf{err}_3^{\sigma}(M) = 5$ and $\mathsf{piv}_3^{\sigma}(M) = 0$. Therefore we find that $\sigma \in \operatorname{Aut} M$.

•

Example 5.21. Consider the lexically ordered matrix $M \in \mathcal{M}_5$ below. Let $\sigma = (1\ 2)$. The recursion tree shown corresponds to the traversal of $U^{(1)}\,\sigma$ by Algorithm 5.4.

$$M = \left(\begin{array}{ccccc} 0\,\vert & 1 & 1 & 1 & 1 \\ 1 & 0\,\vert & 1 & 1\,\vert & 2 \\ 1 & 1 & 0\,\vert & 2\,\vert & 1 \\ 1 & 1 & 2 & 0\,\vert & 2 \\ 1 & 2 & 1 & 2 & 0 \end{array}\right)$$



The shaded areas contain the right cosets which are now also discarded by Algorithm 5.5.

1. We have $\mathsf{lex}_1^{\sigma}(M) = (\{1,3,4\},\{5\})$ and both $\mathsf{tu}_1(M) = \{(1),(1),(1),(\mathbf{1})\}$ and $\mathsf{tu}_1^{\sigma}(M) = \{(1),(1),(1),(\mathbf{2})\}$. Hence $\mathsf{err}_1^{\sigma}(M) = 5$, $\mathsf{piv}_1^{\sigma}(M) = 1$ and $\mathsf{act}_1^{\sigma}(M) = 1$. Therefore only the right cosets $U^{(2)}\,(1\,2)$, $U^{(2)}\,(1\,2\,3)$ and $U^{(2)}\,(1\,2\,4)$ need to be considered because $\mathsf{lex}_1^{\sigma}(M)_{[1]} = \{1,3,4\}$.
2. We have $\mathsf{lex}_2^{\sigma}(M) = (\{3,4\},\{5\})$ and both $\mathsf{tu}_2(M) = \{(1,1),(1,1),(\mathbf{1},2)\}$ and $\mathsf{tu}_2^{\sigma}(M) = \{(1,1),(1,1),(\mathbf{2},1)\}$. Hence $\mathsf{err}_2^{\sigma}(M) = 5$, $\mathsf{piv}_2^{\sigma}(M) = 1$ and $\mathsf{act}_2^{\sigma}(M) = 1$. Therefore we find that only the right cosets $U^{(3)}\,(1\,2)$ and $U^{(3)}\,(1\,2)\,(3\,4)$ need to be considered because $\mathsf{lex}_2^{\sigma}(M)_{[1]} = \{3,4\}$.
3. We have $\mathsf{lex}_3^{\pi}(M) = (\{3\},\{5\})$ and both $\mathsf{tu}_3(M) = \{(1,1,2),(\mathbf{1},2,1)\}$ and $\mathsf{tu}_3^{\pi}(M) = \{(1,1,2),(\mathbf{2},1,2)\}$. Hence $\mathsf{err}_3^{\pi}(M) = 5$, $\mathsf{piv}_3^{\pi}(M) = 1$ and $\mathsf{act}_3^{\pi}(M) = 1$. Therefore we find $(g\,M)_5 > M_5$ for all $g \in U^{(3)}\,\pi$. Hence no such g could ever lead to a counterexample of the minimality of M.
4. We have $\mathsf{lex}_2^{\sigma}(M) = (\{1\},\{4\},\{5\})$, $\mathsf{tu}_2(M) = \{(1,1),(1,\mathbf{1}),(1,2)\}$ and $\mathsf{tu}_2^{\sigma}(M) = \{(1,1),(1,\mathbf{2}),(2,1)\}$. Hence $\mathsf{err}_2^{\sigma}(M) = 4$, $\mathsf{piv}_2^{\sigma}(M) = 2$ and $\mathsf{act}_2^{\sigma}(M) = 1$. Therefore we find that $(g\,M)_4 > M_4$ for all $g \in U^{(2)}\,\sigma$. Hence no such g could ever lead to a counterexample to the minimality of M.

•

5.4.3 Refinement algorithm

In the previous section we introduced several improvements to our canonicity algorithm, at a rather high level of abstraction. These improvements rely on an efficient computation of both lexically order partitions as well as tentative error levels, tentative pivot levels and tentative actions which are required throughout the entire course of the canonicity test. In this section we shift our focus to a detailed, low-level description of these computations.

In general, let $M \in \mathcal{M}_n$ and $i \in \{1,\ldots,n-1\}$. Assume we are at level i in the recursion and Algorithm 5.5 is about to traverse the right coset $U^{(i)}\,\pi$. The changes incorporated into our algorithm require an efficient computation of both $\mathsf{lex}_i^\pi(M)$ as well as $\mathsf{err}_i^\pi(M)$, $\mathsf{piv}_i^\pi(M)$ and $\mathsf{act}_i^\pi(M)$. On the one hand the computation of $\mathsf{lex}_i^\pi(M)$ essentially amounts to sorting the elements $j \in \{(i+1)^\pi, \ldots, n^\pi\}$ according to the lexicographic ordering on the tuples $M_{1^\pi \ldots i^\pi, j}$, or to put it differently, to construct $\mathsf{tu}_i^\pi(M)$. On the other hand the computation of $\mathsf{piv}_i^\pi(M)$, $\mathsf{err}_i^\pi(M)$ and $\mathsf{act}_i^\pi(M)$ essentially amounts to running through the tuples of $\mathsf{tu}_i(M)$ and $\mathsf{tu}_i^\pi(M)$ simultaneously, until we either have encountered two tuples which differ or have considered all such tuples.

Performing the actual sort and at the same time comparing the corresponding tuples would cancel out the benefits we obtain from reducing the number of right cosets we have to consider. Fortunately, in order to compute $\mathsf{lex}_i^\pi(M)$, $\mathsf{err}_i^\pi(M)$, $\mathsf{piv}_i^\pi(M)$ and $\mathsf{act}_i^\pi(M)$ we can exploit the preceding computation of $\mathsf{lex}_{i-1}^\sigma(M)$, $\mathsf{err}_{i-1}^\sigma(M)$, $\mathsf{piv}_{i-1}^\sigma(M)$ and $\mathsf{act}_{i-1}^\sigma(M)$, where σ is the representative of the coset considered one level higher in the recursion. To be able to outline these more efficient computations, we introduce the following theorems:

Theorem 5.4.20. *Let $M \in \mathcal{M}_n$, let ρ, $\varphi \in \mathrm{Sym}(n)$ and $l \in \{1, \ldots, n-1\}$ such that $\varphi \in U^{(l-1)}\rho$. Then the ordered partition $P \in \mathcal{P}(\{1, \ldots, n\})$ defined by $\overset{l,\varphi,M}{\lesssim}$ is a* cell order preserving refinement *of the ordered partition $Q \in \mathcal{P}(\{1, \ldots, n\})$ defined by $\overset{l-1,\rho,M}{\lesssim}$.*

Proof : Because $\varphi \in U^{(l-1)}\rho$, we have $M_{1^\rho \ldots (l-1)^\rho, p} = M_{1^\varphi \ldots (l-1)^\varphi, p}$ and $M_{1^\rho \ldots (l-1)^\rho, q} = M_{1^\varphi \ldots (l-1)^\varphi, q}$. If $[p]_{l,\varphi,M} = [q]_{l,\varphi,M}$, then $M_{1^\varphi \ldots l^\varphi, p} = M_{1^\varphi \ldots l^\varphi, q}$. Hence also $M_{1^\rho \ldots (l-1)^\rho, p} = M_{1^\rho \ldots (l-1)^\rho, q}$ and therefore $[p]_{l-1,\rho,M} = [q]_{l-1,\rho,M}$. Otherwise if $[p]_{l,\varphi,M} \overset{l,\varphi,M}{<} [q]_{l,\varphi,M}$, then $M_{1^\varphi \ldots l^\varphi, p} < M_{1^\varphi \ldots l^\varphi, q}$. We distinguish between the following two cases:

- If $M_{1^\varphi \ldots (l-1)^\varphi, p} = M_{1^\varphi \ldots (l-1)^\varphi, q}$ and $M_{l^\varphi, p} < M_{l^\varphi, q}$, then $M_{1^\rho \ldots (l-1)^\rho, p} = M_{1^\rho \ldots (l-1)^\rho, q}$. Hence we find that $[p]_{l-1,\rho,M} = [q]_{l-1,\rho,M}$.
- If $M_{1^\varphi \ldots (l-1)^\varphi, p} < M_{1^\varphi \ldots (l-1)^\varphi, q}$, then also $M_{1^\rho \ldots (l-1)^\rho, p} < M_{1^\rho \ldots (l-1)^\rho, q}$. Hence we find that $[p]_{l-1,\rho,M} \overset{l-1,\rho,M}{<} [q]_{l-1,\rho,M}$.

Hence in both subcases we find that $[p]_{l-1,\rho,M} \overset{l-1,\rho,M}{\leq} [q]_{l-1,\rho,M}$. ∎

Corollary 5.4.2. *Let $M \in \mathcal{M}_n$, let ρ, $\varphi \in \mathrm{Sym}(n)$ and $l \in \{1, \ldots, n-1\}$ such that $\varphi \in U^{(l-1)}\rho$ and $p, q \in \{(l+1)^\varphi, \ldots, n^\varphi\}$. Then the following relations* *hold:*

- *If* $\mathsf{cell}(\mathsf{lex}_l^{\varphi}(M), p) = \mathsf{cell}(\mathsf{lex}_l^{\varphi}(M), q)$*, then we have* $\mathsf{cell}(\mathsf{lex}_{l-1}^{\rho}(M), p) = \mathsf{cell}(\mathsf{lex}_{l-1}^{\rho}(M), q)$.
- *If* $\mathsf{cell}(\mathsf{lex}_{l-1}^{\rho}(M), p) < \mathsf{cell}(\mathsf{lex}_{l-1}^{\rho}(M), q)$*, then we have* $\mathsf{cell}(\mathsf{lex}_l^{\varphi}(M), p) < \mathsf{cell}(\mathsf{lex}_l^{\varphi}(M), q)$.
- *If* $\mathsf{cell}(\mathsf{lex}_{l-1}^{\rho}(M), p) = \mathsf{cell}(\mathsf{lex}_{l-1}^{\rho}(M), q)$*, then we have* $\mathsf{cell}(\mathsf{lex}_l^{\varphi}(M), p) \leq \mathsf{cell}(\mathsf{lex}_l^{\varphi}(M), q)$ *if and only if* $M_{l^{\varphi},p} \leq M_{l^{\varphi},q}$.

$\diamond$

Each cell of $\mathsf{lex}_l^{\varphi}(M)$ is clearly a subset of a unique cell of $\mathsf{lex}_{l-1}^{\rho}(M)$. Besides that, $\mathsf{lex}_l^{\varphi}(M)$ *preserves* the *cell order* of $\mathsf{lex}_{l-1}^{\rho}(M)$. More precisely, any two distinct cells of $\mathsf{lex}_l^{\varphi}(M)$ which are subsets of two distinct cells of $\mathsf{lex}_{l-1}^{\rho}(M)$ preserve the cell order of those cells. Moreover, we find that if two distinct cells are subsets of the same cell of $\mathsf{lex}_{l-1}^{\rho}(M)$, then their relative cell order is determined by the order of $M_{l^{\varphi},p}$ and $M_{l^{\varphi},q}$ where p and q are respectively elements of those two distinct cells.

Example 5.22. Let $\tau = (1\ 2)$, $\rho = (1\ 2\ 3)$ and $\pi = (1\ 2\ 6)$. Consider the lexically ordered matrix $M \in \mathcal{M}_6$ and some associated partitions $\mathsf{lex}_i^h(M)$ below. Above each cell $\mathsf{lex}_i^h(M)_{[a]}$ we have listed the corresponding tuple $\mathsf{tu}_i^h(M)_{[a]}$.

$$M = \begin{pmatrix} 0 & \vert\, 1 & 1 & 1 & 1 & \vert\, 2 \\ 1 & 0 & \vert\, 1 & \vert\, 2 & 2 & \vert\, 1 \\ 1 & 1 & 0 & \vert\, 2 & 2 & \vert\, 2 \\ 1 & 2 & 2 & 0 & \vert\, 1 & \vert\, 1 \\ 1 & 2 & 2 & 1 & 0 & \vert\, 2 \\ 2 & 1 & 2 & 1 & 2 & 0 \end{pmatrix}$$

$$\begin{aligned} \mathsf{lex}_1^{\tau}(M) &= (\overset{(1)}{\{1,3,6\}}, \overset{(2)}{\{4,5\}}) \\ \mathsf{lex}_2^{\rho}(M) &= (\overset{\binom{1}{1}}{\{\ 1\ \}}, \overset{\binom{1}{2}}{\{\ 6\ \}}, \overset{\binom{2}{2}}{\{4,5\}}) \\ \mathsf{lex}_2^{\pi}(M) &= (\overset{\binom{1}{2}}{\{1,3\}}, \overset{\binom{2}{1}}{\{\ 4\ \}}, \overset{\binom{2}{2}}{\{\ 5\ \}}) \end{aligned}$$

Distinguish between the following cases:

- The partition $\mathsf{lex}_2^{\rho}(M)$ preserves the cell order of $\mathsf{lex}_1^{\tau}(M)$. Moreover, note that $\mathsf{cell}(\mathsf{lex}_2^{\rho}(M), 1) < \mathsf{cell}(\mathsf{lex}_2^{\rho}(M), 6)$ as $M_{2^{\rho},1} < M_{2^{\rho},6}$ while clearly $\mathsf{cell}(\mathsf{lex}_2^{\rho}(M), 4) = \mathsf{cell}(\mathsf{lex}_2^{\rho}(M), 5)$ as $M_{2^{\rho},4} = M_{2^{\rho},5}$.
- The partition $\mathsf{lex}_2^{\pi}(M)$ preserves the cell order of $\mathsf{lex}_1^{\tau}(M)$. Moreover, note

that $\mathsf{cell}(\mathsf{lex}_2^\pi(M),1) = \mathsf{cell}(\mathsf{lex}_2^\pi(M),3)$ as $M_{2^\pi,1} = M_{2^\pi,3}$ while clearly $\mathsf{cell}(\mathsf{lex}_2^\pi(M),4) < \mathsf{cell}(\mathsf{lex}_2^\pi(M),5)$ as $M_{2^\pi,4} < M_{2^\pi,5}$.

•

In the general context of the description of the refinement algorithm in the remainder of this section and the refinement data structure in the next section, it will be convenient to write $\mathsf{lex}_{i-1}^\pi(M)$, $\mathsf{piv}_{i-1}^\pi(M)$, $\mathsf{err}_{i-1}^\pi(M)$ and $\mathsf{act}_{i-1}^\pi(M)$ instead of respectively $\mathsf{lex}_{i-1}^\sigma(M)$, $\mathsf{piv}_{i-1}^\sigma(M)$, $\mathsf{err}_{i-1}^\sigma(M)$ and $\mathsf{act}_{i-1}^\sigma(M)$ because σ and π have the same $i-1$ prefix.

Furthermore, according to Corollary 5.4.2 we may assume that $\mathsf{lex}_{i-1}^\pi(M)$ and $\mathsf{lex}_i^\pi(M)$ are of the form

$$\begin{aligned}\mathsf{lex}_{i-1}^\pi(M) &= \Big(\mathsf{lex}_{i-1}^\pi(M)_{[1]}, \mathsf{lex}_{i-1}^\pi(M)_{[2]}, \ldots, , \mathsf{lex}_{i-1}^\pi(M)_{[r]}\Big) & (5.22)\\ \mathsf{lex}_i^\pi(M) &= \Big(\mathsf{lex}_i^\pi(M)_{[a_1]},\ldots,\mathsf{lex}_i^\pi(M)_{[b_1]}, \mathsf{lex}_i^\pi(M)_{[a_2]},\ldots,\mathsf{lex}_i^\pi(M)_{[b_2]}, \ldots\\ &\qquad \ldots, \mathsf{lex}_i^\pi(M)_{[a_r]},\ldots,\mathsf{lex}_i^\pi(M)_{[b_r]}\Big) & (5.23)\end{aligned}$$

where $\bigcup_{j=a_1}^{b_1} \mathsf{lex}_i^\pi(M)_{[j]} = \mathsf{lex}_{i-1}^\pi(M)_{[1]} \setminus \{i^\pi\}$, $\bigcup_{j=a_k}^{b_k} \mathsf{lex}_i^\pi(M)_{[j]} = \mathsf{lex}_{i-1}^\pi(M)_{[k]}$ and $r = |\mathsf{lex}_{i-1}^\pi(M)|$ for all $k \in \{2,\ldots,r\}$.

Similarly, we may assume that $\mathsf{lex}_{i-1}(M)$ and $\mathsf{lex}_i(M)$ are of the form

$$\begin{aligned}\mathsf{lex}_{i-1}(M) &= \Big(\mathsf{lex}_{i-1}(M)_{[1]}, \mathsf{lex}_{i-1}(M)_{[2]}, \ldots, , \mathsf{lex}_{i-1}(M)_{[s]}\Big) & (5.24)\\ \mathsf{lex}_i(M) &= \Big(\mathsf{lex}_i(M)_{[c_1]},\ldots,\mathsf{lex}_i(M)_{[d_1]}, \mathsf{lex}_i(M)_{[c_2]},\ldots,\mathsf{lex}_i(M)_{[d_2]}, \ldots\\ &\qquad \ldots, \mathsf{lex}_i(M)_{[c_s]},\ldots,\mathsf{lex}_i(M)_{[d_s]}\Big) & (5.25)\end{aligned}$$

where $\bigcup_{j=c_1}^{d_1} \mathsf{lex}_i(M)_{[j]} = \mathsf{lex}_{i-1}(M)_{[1]} \setminus \{i\}$, $\bigcup_{j=c_k}^{d_k} \mathsf{lex}_i(M)_{[j]} = \mathsf{lex}_{i-1}(M)_{[k]}$ and $s = |\mathsf{lex}_{i-1}(M)|$ for all $k \in \{2,\ldots,s\}$.

Lemma 5.4.21. *Let $M \in \mathcal{M}_n$, let $\rho \in \mathrm{Sym}(n)$ and $l \in \{2,\ldots,n-1\}$ such that $\varphi \in U^{(l-1)}\rho$ and $\mathsf{tu}_{l-1}(M)_{[a]} = \mathsf{tu}_{l-1}^\rho(M)_{[b]}$. Assume $\mathsf{lex}_l(M)_{[a']} \subseteq \mathsf{lex}_{l-1}(M)_{[a]}$ and $\mathsf{lex}_l^\varphi(M)_{[b']} \subseteq \mathsf{lex}_{l-1}^\rho(M)_{[b]}$. Then $\mathsf{tu}_l(M)_{[a']} = \mathsf{tu}_l^\varphi(M)_{[b']}$ if and only if $M_{l,p} = M_{l^\varphi,q}$ where $p \in \mathsf{lex}_l(M)_{[a']}$ and $q \in \mathsf{lex}_l^\varphi(M)_{[b']}$*

Proof : Since $\mathsf{lex}_l^\rho(M)$ and $\mathsf{tu}_{l-1}^\rho(M)_{[b]}$ only depends on the $(l-1)$-prefix of ρ and $\varphi \in U^{(l-1)}\rho$, it is sufficient to prove this lemma for $\rho = \varphi$. As $\mathsf{tu}_{l-1}(M)_{[a]} = \mathsf{tu}_{l-1}^\varphi(M)_{[b]}$, $\mathsf{lex}_l(M)_{[a']} \subseteq \mathsf{lex}_{l-1}(M)_{[a]}$, $\mathsf{lex}_l^\varphi(M)_{[b']} \subseteq \mathsf{lex}_{l-1}^\varphi(M)_{[b]}$ we have $M_{1\ldots l-1,p} = M_{1^\varphi\ldots(l-1)^\varphi,q}$. Hence $M_{1\ldots l,p} = M_{1^\varphi\ldots l^\varphi,q}$, or equivalently $\mathsf{tu}_l(M)_{[a']} = \mathsf{tu}_l^\varphi(M)_{[b']}$, if and only if $M_{l,p} = M_{l^\varphi,q}$. ■

Theorem 5.4.22. *Let $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1,\ldots,n-1\}$ such that $\mathsf{err}_{i-1}^\pi(M) > i$. Write $e = \mathsf{err}_{i-1}^\pi(M)$ and $t = \mathsf{cell}(\mathsf{lex}_i(M), e-1)$. If for each $a \in \{1,\ldots,t\}$ we have*

$$|\mathsf{lex}_i(M)_{[a]}| = |\mathsf{lex}_i^\pi(M)_{[a]}| \quad \textit{and} \quad M_{i,q_a} = M_{i^\pi,p_a}$$

with $q_a \in \mathsf{lex}_i(M)_{[a]}$ and $p_a \in \mathsf{lex}_i^\pi(M)_{[a]}$, then $\mathsf{err}_i^\pi(M) = \mathsf{err}_{i-1}^\pi(M)$. Otherwise, let $b \in \{1,\ldots,t\}$ be the minimal element such that

$$|\mathsf{lex}_i(M)_{[b]}| \neq |\mathsf{lex}_i^\pi(M)_{[b]}| \quad \textit{or} \quad M_{i,q_b} \neq M_{i^\pi,p_b}$$

with $q_b \in \mathsf{lex}_i(M)_{[b]}$ and $p_b \in \mathsf{lex}_i^\pi(M)_{[b]}$. Then the following relations hold:

1. *If $M_{i,q_b} \neq M_{i^\pi,p_b}$, then $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[b]})$. Moreover, if $\mathsf{err}_i^\pi(M) < \mathsf{err}_{i-1}^\pi(M)$ then $\mathsf{act}_i^\pi(M) = 1$ if $M_{i,q_b} < M_{i^\pi,p_b}$ and $\mathsf{act}_i^\pi(M) = -1$ otherwise.*

2. *Otherwise $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[b]}) + \min(|\mathsf{lex}_i^\pi(M)_{[b]}|, |\mathsf{lex}_i(M)_{[b]}|)$. Besides, if $\mathsf{err}_i^\pi(M) < \mathsf{err}_{i-1}^\pi(M)$ then $\mathsf{act}_i^\pi(M) = 1$ if $|\mathsf{lex}_i(M)_{[b]}| > |\mathsf{lex}_i^\pi(M)_{[b]}|$ and $\mathsf{act}_i^\pi(M) = -1$ otherwise.*

Proof : Write $s = |\mathsf{lex}_{i-1}(M)|$ and $r = |\mathsf{lex}_{i-1}^\pi(M)|$, then $\mathsf{lex}_{i-1}(M)$ and $\mathsf{lex}_{i-1}^\pi(M)$ correspond to (5.24) and (5.22), respectively. Hence, as a result of Theorem 5.4.8, $\mathsf{tu}_{i-1}(M)$ and $\mathsf{tu}_{i-1}^\pi(M)$ can be uniquely written as

$$\begin{aligned}
\mathsf{tu}_{i-1}(M) &= (\overbrace{\mathsf{tu}_{i-1}(M)_{[1]},\ldots,\mathsf{tu}_{i-1}(M)_{[1]}}^{|\mathsf{lex}_{i-1}(M)_{[1]}|\text{ times}},\ldots,\overbrace{\mathsf{tu}_{i-1}(M)_{[s]},\ldots,\mathsf{tu}_{i-1}(M)_{[s]}}^{|\mathsf{lex}_{i-1}(M)_{[s]}|\text{ times}})\\
\mathsf{tu}_{i-1}^\pi(M) &= (\overbrace{\mathsf{tu}_{i-1}^\pi(M)_{[1]},\ldots,\mathsf{tu}_{i-1}^\pi(M)_{[1]}}^{|\mathsf{lex}_{i-1}^\pi(M)_{[1]}|\text{ times}},\ldots,\overbrace{\mathsf{tu}_{i-1}^\pi(M)_{[r]},\ldots,\mathsf{tu}_{i-1}^\pi(M)_{[r]}}^{|\mathsf{lex}_{i-1}^\pi(M)_{[r]}|\text{ times}}).
\end{aligned}$$

Write $t' = \mathsf{cell}(\mathsf{lex}_i(M), e-1)$. As M is lexically ordered, we have $M_{1\ldots i-1,e-1} = \mathsf{tu}_{i-1}(M)_{e-l} = \mathsf{tu}_{i-1}(M)_{[t']}$. Because $e = \mathsf{err}_{i-1}^\pi(M)$, the first $e-i$ tuples of

$\mathsf{tu}_{i-1}(M)$ and $\mathsf{tu}^{\pi}_{i-1}(M)$ must coincide. Hence $\mathsf{tu}_{i-1}(M)_{[k]} = \mathsf{tu}^{\pi}_{i-1}(M)_{[k]}$ for each $k \in \{1, \dots, t'\}$. Moreover $|\mathsf{lex}_{i-1}(M)_{[j]}| = |\mathsf{lex}^{\pi}_{i-1}(M)_{[j]}|$ for each $j \in \{1, \dots, t'-1\}$. Note that $|\mathsf{lex}_{i-1}(M)_{[t']}|$ needs not necessarily to be equal to $|\mathsf{lex}^{\pi}_{i-1}(M)_{[t']}|$.

According to Corollary 5.4.2 we find that $\mathsf{lex}_i(M)$ and $\mathsf{lex}^{\pi}_i(M)$ correspond to (5.25) and (5.23), respectively. Hence, as a result of Theorem 5.4.8 we find that $\mathsf{tu}_i(M)$ can be uniquely written as

$$\begin{aligned}\mathsf{tu}_i(M) = {} & (\overbrace{\mathsf{tu}_i(M)_{[c_1]},\dots,\mathsf{tu}_i(M)_{[c_1]}}^{|\mathsf{lex}_i(M)_{[c_1]}| \text{ times}}, \dots, \overbrace{\mathsf{tu}_i(M)_{[d_1]},\dots,\mathsf{tu}_i(M)_{[d_1]}}^{|\mathsf{lex}_i(M)_{[d_1]}| \text{ times}}, \dots \\ & \dots, \overbrace{\mathsf{tu}_i(M)_{[c_s]},\dots,\mathsf{tu}_i(M)_{[c_s]}}^{|\mathsf{lex}_i(M)_{[c_s]}| \text{ times}}, \dots, \overbrace{\mathsf{tu}_i(M)_{[d_s]},\dots,\mathsf{tu}_i(M)_{[d_s]}}^{|\mathsf{lex}_i(M)_{[d_s]}| \text{ times}})\end{aligned} \tag{5.26}$$

while $\mathsf{tu}^{\pi}_i(M)$ takes the form

$$\begin{aligned}\mathsf{tu}^{\pi}_i(M) = {} & (\overbrace{\mathsf{tu}^{\pi}_i(M)_{[a_1]},\dots,\mathsf{tu}^{\pi}_i(M)_{[a_1]}}^{|\mathsf{lex}^{\pi}_i(M)_{[a_1]}| \text{ times}}, \dots, \overbrace{\mathsf{tu}^{\pi}_i(M)_{[b_1]},\dots,\mathsf{tu}^{\pi}_i(M)_{[b_1]}}^{|\mathsf{lex}^{\pi}_i(M)_{[b_1]}| \text{ times}}, \dots \\ & \dots, \overbrace{\mathsf{tu}^{\pi}_i(M)_{[a_r]},\dots,\mathsf{tu}^{\pi}_i(M)_{[a_r]}}^{|\mathsf{lex}^{\pi}_i(M)_{[a_r]}| \text{ times}}, \dots, \overbrace{\mathsf{tu}^{\pi}_i(M)_{[b_r]},\dots,\mathsf{tu}^{\pi}_i(M)_{[b_r]}}^{|\mathsf{lex}^{\pi}_i(M)_{[b_r]}| \text{ times}})\end{aligned} \tag{5.27}$$

Note that $t \in \{c_{t'}, \dots, d_{t'}\}$.

First consider the case where $|\mathsf{lex}_i(M)_{[a]}| = |\mathsf{lex}^{\pi}_i(M)_{[a]}|$ and $M_{i,q_a} = M_{i^{\pi},p_a}$ for each $a \in \{1, \dots, t\}$ with $q_a \in \mathsf{lex}_i(M)_{[a]}$ and $p_a \in \mathsf{lex}^{\pi}_i(M)_{[a]}$. Lemma 5.4.21 states that $\mathsf{tu}_i(M)_{[a]} = \mathsf{tu}^{\pi}_i(M)_{[a]}$ for each such a. Hence at least the first $e - i - 1$ tuples of (5.26) and (5.27) must coincide. Using Theorem 5.4.17 we therefore find that $\mathsf{err}^{\pi}_i(M) = \mathsf{err}^{\pi}_{i-1}(M)$.

Otherwise, let $b \in \{1, \dots, t\}$ be the minimal element for which $|\mathsf{lex}_i(M)_{[b]}| \neq |\mathsf{lex}^{\pi}_i(M)_{[b]}|$ or $M_{i,q_b} \neq M_{i^{\pi},p_b}$ with $q_b \in \mathsf{lex}_i(M)_{[b]}$ and $p_b \in \mathsf{lex}^{\pi}_i(M)_{[b]}$. We distinguish between the following cases:

1. If $M_{i,q_b} \neq M_{i^{\pi},p_b}$, then Lemma 5.4.21 states that $\mathsf{tu}_i(M)_{[c]} = \mathsf{tu}^{\pi}_i(M)_{[c]}$ for each $c \in \{1, \dots, b-1\}$. Hence the tuples of (5.26) and (5.27) with

index numbers up to

$$\sum_{k=1}^{b-1} |\mathsf{lex}_i(M)_{[k]}|,$$

or equivalently

$$\min(\mathsf{lex}_i(M)_{[b]}) - i - 1,$$

(as M is lexically ordered), must coincide whereas the next tuple must differ. Hence clearly $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[b]})$. Moreover, if $\mathsf{err}_i^\pi(M) < \mathsf{err}_{i-1}^\pi(M)$ then we find according to Corollary 5.4.1 that $\mathsf{act}_i^\pi(M) = 1$ if $M_{i,q_b} < M_{i^\pi,p_b}$ whereas $\mathsf{act}_i^\pi(M) = -1$ otherwise.

2. If $M_{i,q_b} = M_{i^\pi,p_b}$, then Lemma 5.4.21 states that $\mathsf{tu}_i(M)_{[d]} = \mathsf{tu}_i^\pi(M)_{[d]}$ for each $d \in \{1,\dots,b\}$. Hence the tuples of (5.26) and (5.27) with index numbers up to

$$\sum_{k=1}^{b-1} |\mathsf{lex}_i(M)_{[k]}| + \min(|\mathsf{lex}_i^\pi(M)_{[b]}|, |\mathsf{lex}_i(M)_{[b]}|),$$

or equivalently

$$\min(\mathsf{lex}_i(M)_{[b]}) + \min(|\mathsf{lex}_i^\pi(M)_{[b]}|, |\mathsf{lex}_i(M)_{[b]}|) - i - 1$$

(as M is lexically ordered), must coincide whereas the next tuple must differ. Hence $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[b]}) + \min(|\mathsf{lex}_i^\pi(M)_{[b]}|, |\mathsf{lex}_i(M)_{[b]}|)$. Moreover, if $\mathsf{err}_i^\pi(M) < \mathsf{err}_{i-1}^\pi(M)$ then we find according to Corollary 5.4.1 that $\mathsf{act}_i^\pi(M) = 1$ if $|\mathsf{lex}_i(M)_{[b]}| > |\mathsf{lex}_i^\pi(M)_{[b]}|$ whereas $\mathsf{act}_i^\pi(M) = -1$ otherwise.

■

Example 5.23. Let $\tau = (1\ 2)$, $\rho = (1\ 2\ 3)$ and $\pi = (1\ 2\ 6)$. Consider the lexically ordered matrix $M \in \mathcal{M}_6$ below. Some associated partitions $\mathsf{lex}_i^h(M)$ and tuple sequences $\mathsf{tu}_i^h(M)$ are given aside. Above each cell $\mathsf{lex}_i^h(M)_{[a]}$ we have listed the corresponding tuple $\mathsf{tu}_i^h(M)_{[a]}$.

$$M=\left(\begin{array}{cccccc} 0\,| & 1 & 1 & 1 & 1\,| & 2 \\ 1 & 0\,| & 1\,| & 2 & 2\,| & 1 \\ 1 & 1 & 0\,| & 2 & 2\,| & 2 \\ 1 & 2 & 2 & 0\,| & 1\,| & 1 \\ 1 & 2 & 2 & 1 & 0\,| & 2 \\ 2 & 1 & 2 & 1 & 2 & 0 \end{array}\right) \qquad \begin{array}{ll} \mathsf{lex}_1(M)=(\overset{(1)}{\{2,3,4,5\}},\overset{(2)}{\{6\}}) & \mathsf{tu}_1(M)=((1),(1),(1),(1),(2)) \\ \mathsf{lex}_1^{\tau}(M)=(\overset{(1)}{\{1,3,6\}},\overset{(2)}{\{4,5\}}) & \mathsf{tu}_1^{\tau}(M)=((1),(1),(1),(\mathbf{2}),(2)) \\ \mathsf{lex}_2(M)=(\overset{\binom{1}{1}}{\{3\}},\overset{\binom{1}{2}}{\{4,5\}},\overset{\binom{2}{1}}{\{6\}}) & \mathsf{tu}_2(M)=(\binom{1}{1},\binom{1}{2},\binom{1}{2},\binom{2}{1}) \\ \mathsf{lex}_2^{\rho}(M)=(\overset{\binom{1}{1}}{\{1\}},\overset{\binom{1}{2}}{\{6\}},\overset{\binom{2}{2}}{\{4,5\}}) & \mathsf{tu}_2^{\rho}(M)=(\binom{1}{1},\binom{1}{2},\binom{\mathbf{2}}{2},\binom{2}{2}) \\ \mathsf{lex}_2^{\pi}(M)=(\overset{\binom{1}{2}}{\{1,3\}},\overset{\binom{2}{1}}{\{4\}},\overset{\binom{2}{2}}{\{5\}}) & \mathsf{tu}_2^{\pi}(M)=(\binom{1}{\mathbf{2}},\binom{1}{2},\binom{2}{1},\binom{2}{2}) \end{array}$$

Distinguish between the following cases:

- Because $\mathsf{tu}_1(M)_4 < \mathsf{tu}_1^{\tau}(M)_4$ we have $\mathsf{err}_1^{\tau}(M) = 5$, $\mathsf{piv}_1^{\tau}(M) = 1$ and $\mathsf{act}_1^{\tau}(M) = 1$. Clearly $\rho \in U^{(1)}\tau$ such that $2^{\rho} \in \mathsf{lex}_1^{\tau}(M)_{[1]}$. Moreover $M_{2,3} = M_{2^{\rho},1}$ and $|\mathsf{lex}_2(M)_{[1]}| = |\mathsf{lex}_2^{\rho}(M)_{[1]}|$, while $M_{2,4} = M_{2^{\rho},6}$ and $|\mathsf{lex}_2(M)_{[2]}| > |\mathsf{lex}_2^{\rho}(M)_{[2]}|$. Hence Theorem 5.4.22 states that

$$\mathsf{err}_2^{\rho}(M) = \min(\mathsf{lex}_2(M)_{[2]}) + \min(|\mathsf{lex}_2(M)_{[2]}|, |\mathsf{lex}_2^{\rho}(M)_{[2]}|) = 5.$$

- Because $\mathsf{err}_2^{\rho}(M) = \mathsf{err}_1^{\tau}(M)$, we have $\mathsf{piv}_2^{\rho}(M) = \mathsf{piv}_1^{\tau}(M)$ and $\mathsf{act}_2^{\rho}(M) = \mathsf{act}_1^{\tau}(M)$, according to Theorem 5.4.17 and Corollary 5.4.1. Clearly $\pi \in U^{(1)}\tau$ such that $2^{\pi} \in \mathsf{lex}_1^{\tau}(M)_{[1]}$. Moreover $M_{2,3} < M_{2^{\pi},1}$. Hence Theorem 5.4.22 states that $\mathsf{err}_2^{\pi}(M) = \min(\mathsf{err}_2^{\pi}(M)_{[1]}) = 3$.
- Because $\mathsf{err}_2^{\pi}(M) < \mathsf{err}_1^{\tau}(M)$, we find that $\mathsf{piv}_2^{\pi}(M) = 2$ and $\mathsf{act}_2^{\pi}(M) = 1$, according to Theorem 5.4.22 and Theorem 5.4.17.

●

Theorems 5.4.17 and 5.4.22 as well as Corollaries 5.4.1 and 5.4.2 allow a far more efficient computation of $\mathsf{lex}_i^{\pi}(M)$, $\mathsf{err}_i^{\pi}(M)$, $\mathsf{piv}_i^{\pi}(M)$ and $\mathsf{act}_i^{\pi}(M)$ based on the preceding computations of $\mathsf{lex}_{i-1}^{\pi}(M)$, $\mathsf{piv}_{i-1}^{\pi}(M)$, $\mathsf{err}_{i-1}^{\pi}(M)$ and $\mathsf{act}_{i-1}^{\pi}(M)$. The complete refinement algorithm is described in Algorithm 5.6. Recall that we assume that $\mathsf{lex}_{i-1}^{\pi}(M)$, $\mathsf{lex}_i^{\pi}(M)$, $\mathsf{lex}_{i-1}(M)$ and $\mathsf{lex}_i(M)$ correspond to (5.22), (5.24), (5.23) and (5.25) respectively. Note that both $\mathsf{lex}_{i-1}(M)$ and $\mathsf{lex}_i(M)$ can already be computed at the outset of the canonicity algorithm. Moreover, we may assume that $\mathsf{err}_{i-1}^{\pi}(M) > i$, after all, if $\mathsf{err}_{i-1}^{\pi}(M) = i$, then the canonicity test would already have been pruned higher up in the recursion.

Write $e = \mathsf{err}^{\pi}_{i-1}(M)$, $t' = \mathsf{cell}(\mathsf{lex}_{i-1}(M), e-1)$ and $t = \mathsf{cell}(\mathsf{lex}_{i}(M), e-1)$. The refinement algorithm itself is divided into two major parts:

- The method **refine** (π, i) subsequently refines

$$\begin{aligned}\mathsf{lex}^{\pi}_{i-1}(M)_{[1]} \setminus \{i^{\pi}\}, \mathsf{lex}^{\pi}_{i-1}(M)_{[2]}, \ldots \\ , \ldots, \mathsf{lex}^{\pi}_{i-1}(M)_{[t'-1]}, \mathsf{lex}^{\pi}_{i-1}(M)_{[t']}\end{aligned} \tag{5.28}$$

  According to Corollary 5.4.2, we can refine each entry of (5.28) by simply sorting each of their respective elements j according to their corresponding matrix entries $M_{i^{\pi},j}$ (❷ and ❸ in Algorithm 5.6). The computation of $\mathsf{err}^{\pi}_{i}(M)$, $\mathsf{piv}^{\pi}_{i}(M)$ and $\mathsf{act}^{\pi}_{i}(M)$ runs parallel to the systematic refinement of (5.28). After each refinement of either $\mathsf{lex}^{\pi}_{i-1}(M)_{[k]} \setminus \{i^{\pi}\}$ with $k=1$ or $\mathsf{lex}^{\pi}_{i-1}(M)_{[k]}$ with $k \in \{2, \ldots, t'\}$, we systematically invoke the function **checkRefined**(π, i, k) to check whether

$$|\mathsf{lex}_{i}(M)_{[j]}| = |\mathsf{lex}^{\pi}_{i}(M)_{[j]}| \quad \text{and} \quad M_{i,q_j} = M_{i^{\pi},p_j}$$

  for each $j \in \{a_k, \ldots, b_k\}$, independent of the choice of $q_j \in \mathsf{tu}_i(M)_{[j]}$ and $p_j \in \mathsf{tu}^{\pi}_{i}(M)_{[j]}$. If not the function **checkRefined**(π, i, k) returns false, causing the systematic refinement of (5.28) to terminate.

  If the refinement of (5.28) is not terminated prematurely, then

$$|\mathsf{lex}_{i}(M)_{[l]}| = |\mathsf{lex}^{\pi}_{i}(M)_{[l]}| \quad \text{and} \quad \mathsf{tu}_i(M)_{[l]} = \mathsf{tu}^{\pi}_{i}(M)_{[l]}$$

  for each $l \in \{1, \ldots, b_{t'} = t\}$. Hence $\mathsf{err}^{\pi}_{i}(M) = \mathsf{err}^{\pi}_{i-1}(M)$, according to Theorem 5.4.22. Combining Theorem5.4.17 and Corollary 5.4.1 we therefore find that $\mathsf{piv}^{\pi}_{i}(M) = \mathsf{piv}^{\pi}_{i-1}(M)$ and $\mathsf{act}^{\pi}_{i}(M) = \mathsf{act}^{\pi}_{i-1}(M)$ (❶ in Algorithm 5.6). Otherwise, if the refinement of (5.28) is terminated prematurely, both $\mathsf{piv}^{\pi}_{i}(M)$, $\mathsf{err}^{\pi}_{i}(M)$ and $\mathsf{act}^{\pi}_{i}(M)$ are determined by the call to the function **checkRefined** which causes the refinement to terminate.

- The function **checkRefined**(π, i, k) with $k \in \{1, \ldots, t'\}$ checks whether

$$|\mathsf{lex}_{i}(M)_{[j]}| = |\mathsf{lex}^{\pi}_{i}(M)_{[j]}| \quad \text{and} \quad M_{i,q_j} = M_{i^{\pi},p_j} \tag{5.29}$$

  for each $j \in \{a_k, \ldots, b_k\}$. If so, true is returned; otherwise false is returned and both $\mathsf{piv}^{\pi}_{i}(M)$, $\mathsf{err}^{\pi}_{i}(M)$ and $\mathsf{act}^{\pi}_{i}(M)$ are computed. By the structure of the refinement algorithm we have

$$|\mathsf{lex}_{i}(M)_{[l]}| = |\mathsf{lex}^{\pi}_{i}(M)_{[l]}| \quad \text{and} \quad M_{i,q_l} = M_{i^{\pi},p_l}$$

for each $l \in \{a_1, \ldots, b_{k-1}\}$ and therefore $c_k = a_k$. In order to check (5.29) we run through

$$\mathsf{lex}_i(M)_{[c_k]}, \ldots, \mathsf{lex}_i(M)_{[d_k]} \quad \text{and} \quad \mathsf{lex}_i^\pi(M)_{[a_k]}, \ldots, \mathsf{lex}_i^\pi(M)_{[b_k]}$$

simultaneously. For each such pair of cells $\mathsf{lex}_i(M)_{[j]}$ and $\mathsf{lex}_i^\pi(M)_{[j]}$ with $j \in \{a_k, \ldots, b_k\}$, we distinguish between the following cases:

1. If $M_{i,q_j} \neq M_{i^\pi,p_j}$ (❹ in Algorithm 5.6) then Theorem 5.4.22 states that $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[j]})$. Clearly $\mathsf{err}_i^\pi(M) < \mathsf{err}_{i-1}^\pi(M)$. Hence we find according to Theorem 5.4.17 that $\mathsf{piv}_i^\pi(M) = i$. Moreover Theorem 5.4.22 states that $\mathsf{act}_i^\pi(M) = 1$ if $M_{i,q_j} < M_{i^\pi,p_j}$ whereas $\mathsf{act}_i^\pi(M) = -1$ otherwise. Note that false is returned so that the refinement of (5.28) terminates.
2. If $M_{i,q_j} = M_{i^\pi,p_j}$ and $|\mathsf{lex}_i(M)_{[j]}| \neq |\mathsf{lex}_i^\pi(M)_{[j]}|$ (❺ and ❻ in Algorithm 5.6) then Theorem 5.4.22 states that

   $$\mathsf{err}_i^\pi(M) = \begin{cases} \min(\mathsf{lex}_i(M)_{[j]}) + |\mathsf{lex}_i(M)_{[j]}|, & \text{if } |\mathsf{lex}_i(M)_{[j]}| < |\mathsf{lex}_i^\pi(M)_{[j]}|, \\ \min(\mathsf{lex}_i(M)_{[j]}) + |\mathsf{lex}_i^\pi(M)_{[j]}|, & \text{if } |\mathsf{lex}_i(M)_{[j]}| > |\mathsf{lex}_i^\pi(M)_{[j]}|. \end{cases}$$

   If $\mathsf{err}_i^\pi(M) < \mathsf{err}_{i-1}^\pi(M)$, then we find according to Theorems 5.4.22 and 5.4.17 that $\mathsf{piv}_i^\pi(M) = i$ and

   $$\mathsf{act}_i^\pi(M) = \begin{cases} -1 & \text{if } |\mathsf{lex}_i(M)_{[j]}| < |\mathsf{lex}_i^\pi(M)_{[j]}|, \\ \;\;1 & \text{if } |\mathsf{lex}_i(M)_{[j]}| > |\mathsf{lex}_i^\pi(M)_{[j]}|. \end{cases}$$

   Otherwise, if $\mathsf{err}_i^\pi(M) = \mathsf{err}_{i-1}^\pi(M)$, then we find according to Theorem 5.4.17 and Corollary 5.4.1 that $\mathsf{piv}_i^\pi(M) = \mathsf{piv}_{i-1}^\pi(M)$ and $\mathsf{act}_i^\pi(M) = \mathsf{act}_{i-1}^\pi(M)$ (❶ in Algorithm 5.6). Note that false is returned so that the refinement of (5.28) terminates.
3. If $|\mathsf{lex}_i(M)_{[j]}| = |\mathsf{lex}_i^\pi(M)_{[j]}|$, then the next pair of cells $\mathsf{lex}_i(M)_{[j+1]}$ and $\mathsf{lex}_i^\pi(M)_{[j+1]}$ is considered when $j < b_k$; otherwise (5.29) holds and true is returned.

5.4.4 Refinement data structure

Throughout the canonicity test on M, Algorithm 5.6 must keep track of information about lexically ordered image partitions, tentative error levels, ...

Algorithm 5.6 Refinement algorithm

```
method refine ($\pi \in \mathrm{Sym}(n)$, $i$ : int)
 1: $k \leftarrow 1$, $\mathsf{err}_i^\pi(M) \leftarrow \mathsf{err}_{i-1}^\pi(M)$
 2: $\mathsf{piv}_i^\pi(M) \leftarrow \mathsf{piv}_{i-1}^\pi(M)$
 3: $\mathsf{act}_i^\pi(M) \leftarrow \mathsf{act}_{i-1}^\pi(M)$                          ❶
 4: compute $\mathsf{lex}_i^\pi(M)_{[a_k]}, \ldots, \mathsf{lex}_i^\pi(M)_{[b_k]}$
            from $\mathsf{lex}_{i-1}^\pi(M)_{[k]} \setminus \{i^\pi\}$                ❷
 5: while (checkRefined ($\pi$, $i$, $k$) $\wedge$
              $\max(\mathsf{lex}_{i-1}(M)_{[k]}) < \mathsf{err}_{i-1}^\pi(M) - 1$) do
 6:   $k \leftarrow k+1$
 7:   compute $\mathsf{lex}_i^\pi(M)_{[a_k]}, \ldots, \mathsf{lex}_i^\pi(M)_{[b_k]}$
              from $\mathsf{lex}_{i-1}^\pi(M)_{[k]}$                                ❸

function checkRefined ($\pi \in \mathrm{Sym}(n)$, $i$, $k$ : int) : boolean
 1: for $j \leftarrow a_k, \ldots, b_k$ do
 2:   let $p_j \in \mathsf{lex}_i(M)_{[j]}$,  $q_j \in \mathsf{lex}_i^\pi(M)_{[j]}$
 3:   if $M_{i,p_j} \neq M_{i^\pi,q_j}$ then                                        ❹
 4:     $\mathsf{err}_i^\pi(M) \leftarrow \min(\mathsf{lex}_i(M)_{[j]})$, $\mathsf{piv}_i^\pi(M) \leftarrow i$
 5:     if $M_{i,p_j} < M_{i^\pi,q_j}$ then
 6:       $\mathsf{act}_i^\pi(M) = 1$
 7:     else
 8:       $\mathsf{act}_i^\pi(M) = -1$
 9:     return false
10:   else if $|\mathsf{lex}_i(M)_{[j]}| < |\mathsf{lex}_i^\pi(M)_{[j]}|$ then                 ❺
11:     $\mathsf{err}_i^\pi(M) \leftarrow \min(\mathsf{lex}_i(M)_{[j]}) + |\mathsf{lex}_i(M)_{[j]}|$
12:     if $\mathsf{err}_i^\pi(M) \neq \mathsf{err}_{i-1}^\pi(M)$ then
13:       $\mathsf{piv}_i^\pi(M) \leftarrow i$, $\mathsf{act}_i^\pi(M) \leftarrow -1$
14:     return false
15:   else if $|\mathsf{lex}_i(M)_{[j]}| > |\mathsf{lex}_i^\pi(M)_{[j]}|$ then                 ❻
16:     $\mathsf{err}_i^\pi(M) \leftarrow \min(\mathsf{lex}_i(M)_{[j]}) + |\mathsf{lex}_i^\pi(M)_{[j]}|$
17:     if $\mathsf{err}_i^\pi(M) \neq \mathsf{err}_{i-1}^\pi(M)$ then
18:       $\mathsf{piv}_i^\pi(M) \leftarrow i$, $\mathsf{act}_i^\pi(M) \leftarrow 1$
19:     return false
20: return true
```

for every recursion level separately. Besides, each lexically ordered partition should be available already at the outset of the canonicity test rather than be recomputed each time. To perform successive refinements of lexically ordered partitions as well as lexically ordered image partitions, we construct an explicit data structure which maintains these partitions in such a way that it allows a more straightforward (and at the same more efficient) version of the refinement algorithm.

This *refinement data structure* consists of a d-ary tree where d is equal to the maximum value of the matrix entries of M. An array lex is used to maintain certain nodes of this d-ary tree. If we reconsider the general case where we are at level i in the recursion and the canonicity algorithm is about to traverse the right coset $U^{(i)}\pi$, then the element $\mathsf{lex}[i-1]$ at index $i-1$ in the array lex consists of a dynamic array whose element at index a is denoted by $\mathsf{lex}[i-1][a]$. Each such element $\mathsf{lex}[i-1][a]$ corresponds to a unique node in the d-ary tree and contains the following information:

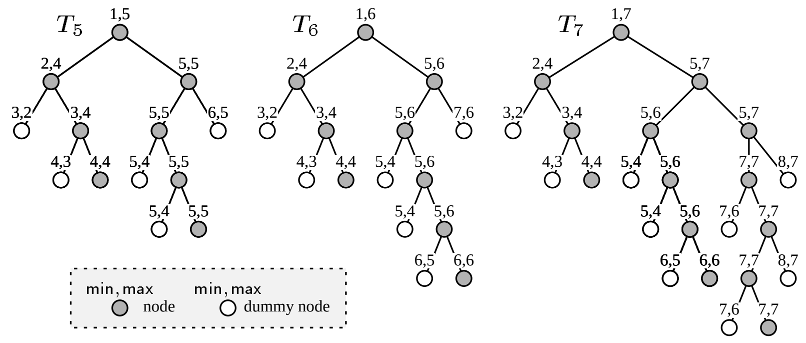
- The minimum and maximum of $\mathsf{lex}_{i-1}(M)_{[a]}$, denoted by min and max, respectively. These values uniquely determine the elements of $\mathsf{lex}_{i-1}(M)_{[a]}$ (cf. Lemma 5.4.9).
- An array ref of d references. If $a = 1$ and $|\mathsf{lex}_{i-1}(M)_{[a]}| = 1$, then $\mathsf{ref}[l]$ is a *null*-reference. Otherwise, if there exists an element $b \in \{1, \ldots, |\mathsf{lex}_i(M)|\}$ such that
$$\mathsf{lex}_i(M)_{[b]} \subseteq \mathsf{lex}_{i-1}(M)_{[a]} \quad \text{and} \quad M_{i,p} = l$$
with $p \in \mathsf{lex}_i(M)_{[b]}$, then $\mathsf{ref}[l]$ refers to the node $\mathsf{lex}[i][b]$. If no such b exists then $\mathsf{ref}[l]$ refers to a *dummy* leaf node for which $\mathsf{ref}[l].\mathsf{min} = \mathsf{ref}[l].\mathsf{max} + 1$ and
$$\mathsf{ref}[l].\mathsf{min} = \begin{cases} \mathsf{ref}[l-1].\mathsf{max} + 1, & \text{if } l > 1, \\ \mathsf{ref}[l+1].\mathsf{min} - 1, & \text{if } l < d. \end{cases}$$
- A dynamic array denoted by img. This array contains the elements of $\mathsf{lex}^{\pi}_{i-1}(M)_{[a]}$ in increasing order if and only if
$$a \leq \mathsf{cell}(\mathsf{lex}_{i-1}(M), \mathsf{err}^{\pi}_{i-1}(M) - 1).$$
We write $|\mathsf{img}|$ for the number of elements of img.

Note that the values of min and max together with the references in the array ref can already be computed at the outset of the canonicity algorithm, while the contents of img depend on the actual right coset we consider at level i in the recursion. For each node we set size $=$ max $-$ min $+ 1$. Hence size $= 0$ if and only if the corresponding node is a dummy leaf node.

Example 5.24. Consider the lexically ordered matrix $M \in \mathcal{M}_7$ below.

$$M = \left(\begin{array}{ccccccccccccc}
0 & | & 1 & & 1 & & 1 & | & 2 & & 2 & & 2 \\
1 & & 0 & | & 2 & & 2 & | & 1 & & 1 & | & 2 \\
1 & & 2 & & 0 & | & 2 & | & 2 & & 2 & | & 1 \\
1 & & 2 & & 2 & & 0 & | & 2 & & 2 & | & 2 \\
2 & & 1 & & 2 & & 2 & & 0 & | & 2 & | & 1 \\
2 & & 1 & & 2 & & 2 & & 2 & & 0 & | & 2 \\
2 & & 2 & & 1 & & 2 & & 1 & & 2 & & 0
\end{array}\right)$$

The binary trees T_5, T_6 and T_7 shown, correspond to the refinement data structure of respectively M_5, M_6 and M_7. For each *internal* node, the *left child* refers to ref[1] and the *right child* refers to ref[2].



Note that the array lex in which all non-dummy nodes are stored is not shown. Although we can easily envision this array. More precisely, if we pass through all non-dummy nodes at a given depth l in the binary tree from left to right, then the b-th non-dummy node corresponds to lex$[l][b]$. Note that the min-value of a dummy node is equal to the min-value of the next non-dummy node on the same level in the tree. Similarly the max-value is equal to the max-value of the previous non-dummy node. It is worth noting that T_5, T_6 and T_7 share a lot of structure. •

Algorithm 5.7 Refinement algorithm

```
method refine($\pi \in \mathrm{Sym}(n)$, $i$ : int)
1:  $k \leftarrow 1$, $\mathsf{err}_i^\pi(M) \leftarrow \mathsf{err}_{i-1}^\pi(M)$
2:  $\mathsf{piv}_i^\pi(M) \leftarrow \mathsf{piv}_{i-1}^\pi(M)$, $\mathsf{act}_i^\pi(M) \leftarrow \mathsf{act}_{i-1}^\pi(M)$
3:  distribute $\mathsf{lex}[i-1][k].\mathsf{img} \setminus \{i^\pi\}$
        among children of $\mathsf{lex}[i-1][k]$                    ①
4:  while (checkRefined ($\pi$, $i$, $k$) $\wedge$
        $\mathsf{lex}[i-1][k].\mathsf{max} < \mathsf{err}_{i-1}^\pi(M) - 1$) do
5:    $k \leftarrow k+1$
6:    distribute $\mathsf{lex}[i-1][k].\mathsf{img}$
          among children of $\mathsf{lex}[i-1][k]$                  ②

function checkRefined ($\pi \in \mathrm{Sym}(n)$, $i$, $k$ : int) : boolean
1:  for $j \leftarrow 1, \ldots, d$ do
2:    $c_j \leftarrow \mathsf{lex}[i-1][k].\mathsf{ref}[j]$
3:    if $c_j.\mathsf{size} < |c_j.\mathsf{img}|$ then                         ③
4:      $\mathsf{err}_i^\pi(M) \leftarrow c_j.\mathsf{min} + c_j.\mathsf{size}$
5:      if $\mathsf{err}_i^\pi(M) \neq \mathsf{err}_{i-1}^\pi(M)$ then
6:        $\mathsf{piv}_i^\pi(M) = i$
7:        $\mathsf{act}_i^\pi(M) = -1$
8:      return false
9:    else if $c_j.\mathsf{size} > |c_j.\mathsf{img}|$ then                    ④
10:     $\mathsf{err}_i^\pi(M) \leftarrow c_j.\mathsf{min} + |c_j.\mathsf{img}|$
11:     if $\mathsf{err}_i^\pi(M) \neq \mathsf{err}_{i-1}^\pi(M)$ then
12:       $\mathsf{piv}_i^\pi(M) = i$
13:       $\mathsf{act}_i^\pi(M) = 1$
14:     return false
15: return true
```

Algorithm 5.7 reformulates the general refinement algorithm as described in Algorithm 5.6, in terms of the refinement data structure outlined above. Recall that we assume that $\mathsf{lex}^{\pi}_{i-1}(M)$, $\mathsf{lex}^{\pi}_{i}(M)$, $\mathsf{lex}_{i-1}(M)$ and $\mathsf{lex}_{i}(M)$ correspond to (5.22), (5.24), (5.23) and (5.25), respectively. The differences between Algorithm 5.7 and Algorithm 5.6 are then in essence the following:

1. The method **refine**(π, i) subsequently refines

$$\mathsf{lex}^{\pi}_{i-1}(M)_{[1]} \setminus \{i^{\pi}\}, \mathsf{lex}^{\pi}_{i-1}(M)_{[2]}, \ldots \\ \ldots, \mathsf{lex}^{\pi}_{i-1}(M)_{[t'-1]}, \mathsf{lex}^{\pi}_{i-1}(M)_{[t']}$$

   by sorting each of their respective elements j according to their corresponding matrix entries $M_{i^{\pi},j}$. The refinement data structure provides us with a rather straightforward method to sort these elements. Indeed, let $k \in \{2, \ldots, t'\}$, then sorting the elements $j \in \mathsf{lex}^{\pi}_{i-1}(M)_{[k]}$ according to $M_{i^{\pi},j}$ simply amounts to scanning through the elements of $\mathsf{lex}[i-1][k].\mathsf{img}$ and distributing them among the children of $\mathsf{lex}[i-1][k]$, that is, each element j of $\mathsf{lex}[i-1][k].\mathsf{img}$ is appended to the array img of its child $\mathsf{lex}[i-1][k].\mathsf{ref}[M_{i^{\pi},j}]$ (cf. ② in Algorithm 5.7). Note that the refinement of $\mathsf{lex}^{\pi}_{i-1}(M)_{[1]} \setminus \{i^{\pi}\}$ is slightly different. All elements of $\mathsf{lex}[i-1][1].\mathsf{img}$, except for i^{π}, are distributed among the children of $\mathsf{lex}[i-1][1]$ (cf. ① in Algorithm 5.7).

2. The function **checkRefined**(i, k) checks whether

$$|\mathsf{lex}_{i}(M)_{[j]}| = |\mathsf{lex}^{\pi}_{i}(M)_{[j]}| \quad \text{and} \quad M_{i,q_j} = M_{i^{\pi},p_j} \tag{5.30}$$

   for each $j \in \{a_k, \ldots, b_k\}$. If so, true is returned; otherwise false is returned and both $\mathsf{piv}^{\pi}_{i}(M)$, $\mathsf{err}^{\pi}_{i}(M)$ and $\mathsf{act}^{\pi}_{i}(M)$ are computed. Further recall that by the structure of the refinement algorithm we have

$$|\mathsf{lex}_{i}(M)_{[l]}| = |\mathsf{lex}^{\pi}_{i}(M)_{[l]}| \quad \text{and} \quad M_{i,q_l} = M_{i^{\pi},p_l} \tag{5.31}$$

   for each $l \in \{a_1, \ldots, b_{k-1}\}$. In terms of the refinement data structure, (5.30) is equivalent to checking whether

$$\mathsf{lex}[i-1][k].\mathsf{ref}[j].\mathsf{size} = |\mathsf{lex}[i-1][k].\mathsf{ref}[j].\mathsf{img}| \tag{5.32}$$

   for each $j \in \{1, \ldots, d\}$, that is the children of $\mathsf{lex}[i-1][k]$. After all, the x-th non-dummy child corresponds to $\mathsf{lex}_{i}(M)_{[a_k+x-1]}$, while the y-th child with $|\mathsf{img}| > 0$ corresponds to $\mathsf{lex}_i^\pi(M)_{[a_k+y-1]}$. In order to check (5.32) we now subsequently run through

$$\mathsf{lex}[i-1][k].\mathsf{ref}[1],\ \mathsf{lex}[i-1][k].\mathsf{ref}[2], \ldots,\ \mathsf{lex}[i-1][k].\mathsf{ref}[d] \tag{5.33}$$

until we either encounter a child of $\mathsf{lex}[i-1][k]$ such that its value size differs from the size of its associated array img or we have considered each child of $\mathsf{lex}[i-1][k]$. Assume that we already have considered the first $l-1$ children of $\mathsf{lex}[i][k]$, among which are z non-dummy nodes with $\mathsf{size} = |\mathsf{img}|$ (and thus $l-1-z$ dummy nodes with $|\mathsf{img}| = 0$). So for the l-th child of $\mathsf{lex}[i][k]$, in this context abbreviated by c_l, we distinguish between the following cases independent of the choice of $q_{a_k+z} \in \mathsf{lex}_i(M)_{[a_k+z]}$ and $p_{a_k+z} \in \mathsf{lex}_i(M)_{[a_k+z]}$:

- If $c_l.\mathsf{size} < |c_l.\mathsf{img}|$ then we consider the following subcases (③ in Algorithm 5.7):
  - If $c_l.\mathsf{size} \neq 0$, then $|\mathsf{lex}_i(M)_{[a_k+z]}| < |\mathsf{lex}_i^\pi(M)_{[a_k+z]}|$ and $M_{i,q_{a_k+z}} = M_{i^\pi,p_{a_k+z}}$ (❺ Algorithm 5.6) and therefore we have
  $$\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[a_k+z]}) + |\mathsf{lex}_i(M)_{[a_k+z]}|.$$
  - If $c_l.\mathsf{size} \neq 0$, then $M_{i,q_{a_k+z}} < M_{i^\pi,p_{a_k+z}}$ (❹ Algorithm 5.6) and therefore we have $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[a_k+z]})$.

  Clearly, in both subcases we find that $\mathsf{err}_i^\pi(M) = c_l.\mathsf{min} + c_l.\mathsf{size}$.
- If $c_l.\mathsf{size} > |c_l.\mathsf{img}|$, then we consider the following subcases (④ in Algorithm 5.7):
  - If $|c_l.\mathsf{img}| = 0$, then we have $|\mathsf{lex}_i(M)_{[a_k+z]}| > |\mathsf{lex}_i^\pi(M)_{[a_k+z]}|$ and $M_{i,q_{a_k+z}} = M_{i^\pi,p_{a_k+z}}$ (cf. ❻ Algorithm 5.6) and therefore we have $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[a_k+z]}) + |\mathsf{lex}_i^\pi(M)_{[a_k+z]}|$.
  - If $|c_l.\mathsf{img}| = 0$, then $M_{i,q_{a_k+z}} < M_{i^\pi,p_{a_k+z}}$ (❹ Algorithm 5.6) and therefore we have $\mathsf{err}_i^\pi(M) = \min(\mathsf{lex}_i(M)_{[a_k+z]})$.

  Clearly in both subcases we find that $\mathsf{err}_i^\pi(M) = c_l.\mathsf{min} + |c_l.\mathsf{img}|$.
- If $c_l.\mathsf{size} = |c_l.\mathsf{img}|$, then we consider the following subcases:
  - If $c_l.\mathsf{size} \neq 0$, then we have $|\mathsf{lex}_i(M)_{[a_k+z]}| < |\mathsf{lex}_i^\pi(M)_{[a_k+z]}|$ and $M_{i,q_{a_k+z}} = M_{i^\pi,p_{a_k+z}}$.
  - If $c_l.\mathsf{size} = 0$, then in this particular case c_l neither identifies $\mathsf{lex}_i(M)_{[a_k+z]}$ nor $\mathsf{lex}_i^\pi(M)_{[a_k+z]}$.

  Clearly if $l < d$, we find in both subcases that we should continue with the next child of $\mathsf{lex}[i-1][k]$.

5.4.5 Analysis and empirical data

The effectiveness of our refinement technique is examined by comparing data obtained from the orderly generation of the same classes of graphs as in Section 5.2.3. In Table 5.4, Algorithm 5.4 and 5.5 are compared. The total number of graphs along with the total number of permutations which are checked during all executed canonicity tests is given. The empirical data illustrates that our refinement technique systematically reduces the number of permutations which are checked during orderly generation.

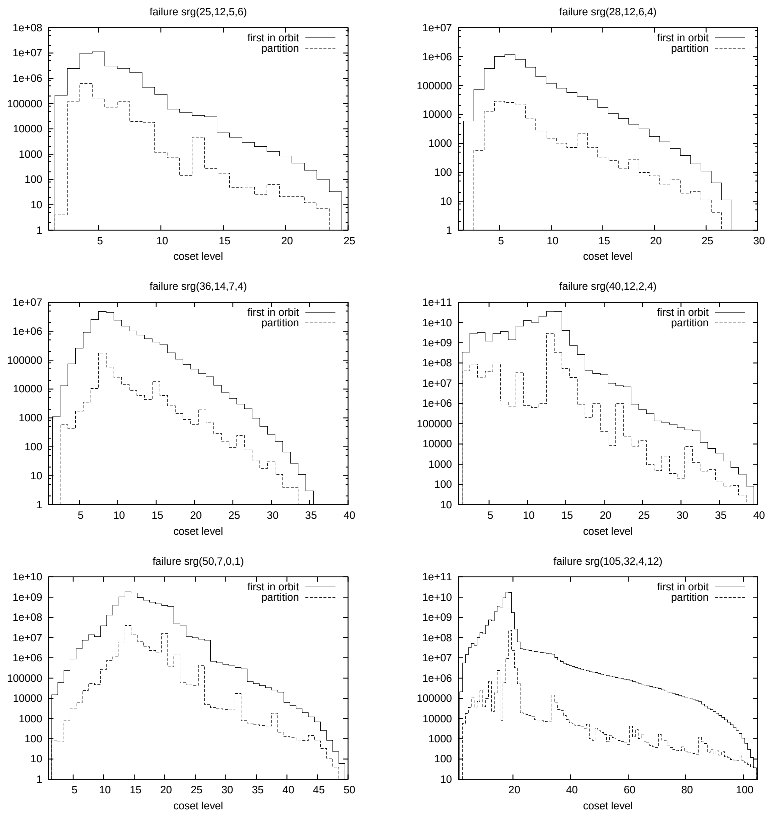
In Figure 5.6 we represent the frequency of occurrence of error levels among all canonicity tests executed during orderly generation. For each parameter set, we can observe a systematic decrease in the number of error levels. This reduction is most apparent for error levels with a large distance between error levels and corresponding average pivot levels as outlined in Figure 5.5. Clearly, our refinement technique improves the performance of the partial permutation criterion by reducing its tendency to repeatedly prune the recursion tree in different parts of the recursion for exactly the same reason (cf. Section 5.4.1).

5.5 Minimality and automorphism group of the leading principal submatrices

The improvements introduced in Section 5.4 all stem from the restriction that the canonicity algorithm takes a *lexically ordered matrix* $M \in \mathcal{M}_n$ as input. In this section we will focus on two additional restrictions on M which allow us to further improve our canonicity algorithm. Our canonicity algorithm is repeatedly applied during the course of an orderly generation algorithm. Hence when we check the canonicity of M, each leading principal submatrix of M is known to be in column order canonical form. Indeed, each leading principal submatrix of M_n must be *minimal*; otherwise the generation algorithm would already have been pruned. On the other hand, the minimality of each leading principal submatrix of M guarantees us that a *strong generating set* for the *automorphism group* of each such submatrix has been obtained.

					Algorithm 5.4	Algorithm 5.5
v	k	λ	μ	$\exists$	No. perm. checked	No. perm checked
5	2	0	1	1	40	15
9	4	1	2	1	457	123
10	3	0	1	1	721	161
	6	3	4		1 214	286
13	6	2	3	1	5 643	737
15	6	1	3	1	7 906	1 172
	8	4	4		12 224	1 731
16	5	0	2	1	21 687	2 762
	10	6	6		36 090	4 530
16	6	2	2	2	23 507	2 953
	9	4	6		47 539	5 600
17	8	3	4	1	41 109	3 751
21	10	3	6	1	75 091	6 966
	10	5	4		14 4207	14 205
25	8	3	2	1	458 845	35 626
	16	9	12		2 654 299	211 670
25	12	5	6	15	34 546 360	2 027 435
26	10	3	4	10	12 714 925	714 234
	15	8	9		257 672 546	18 139 933
27	10	1	5	1	1 289 795	95 807
	16	10	8		3 281 897	231 653
28	12	6	4	4	4 828 422	338 610
	15	6	10		4 068 141	256 250
29	14	6	7	41	3 291 720 746	203 348 967
36	10	4	2	1	23 754 480	1 194 185
	25	16	20		143 366 816	9 066 020
36	14	4	6	180	30 634 602 941	1 415 391 817
36	14	7	4	1	22 069 825	1 114 843
	21	10	15		10 334 741	524 654
40	12	2	4	28	154 334 556 518	8 787 084 036
45	16	8	4	1	305 971 694	15 685 156
	28	15	21		162 721 707	6 755 242
50	7	0	1	1	9 016 393 600	331 518 926
105	32	4	12	1	58 191 699 167	592 209 059

Tabel 5.4: Comparison of algorithm 5.4 and 5.5 used in an orderly algorithm for strongly regular graphs



Figuur 5.6: Frequency of error levels for all canonicity tests executed during a single generation of strongly regular graphs.

5.5.1 Minimality of the leading principal submatrices

Let us first consider the implications of the fact that each leading principal submatrix of M is minimal.

Lemma 5.5.1. *Let $M \in \mathcal{M}_n$ and let M_{n-1} be in canonical form. Let $l \in \{1,\ldots,n-1\}$ and $\pi \in \mathrm{Sym}(n)$ such that $i^\pi \neq n$ for each $i \in \{1,\ldots,l\}$, then $(\pi M)_l \geq M_l$.*

Proof : Let $g' = \pi\,(n^\pi\ n)$ (restricted to $\{1,\ldots,n-1\}$), then $g' \in \mathrm{Sym}(n-1)$ and g' and π have the same l-prefix. It follows that $(\pi M)_l = (g' M_{n-1})_l \geq M_l$, because M_{n-1} is in canonical form. ■

Lemma 5.5.2. *Let $M \in \mathcal{M}_n$ be lexically ordered and let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1,\ldots,n-2\}$ such that $h \in U^{(i)}\pi$ and $i+1 < \mathsf{err}_i^\pi(M) \leq n$. Write $e = \mathsf{err}_i^\pi(M)$. If $(h\,M)_j = M_j$ with $j \in \{i+1,\ldots,e-1\}$, then for each $l \in \{i+1,\ldots,j\}$ we have*

$$M_{1\ldots i,j} = \mathsf{tu}_i(M)_{j-i} \quad \text{and} \quad (h\,M)_{1\ldots i,j} = \mathsf{tu}_i^\pi(M)_{j-i}.$$

Proof : Since $\mathsf{err}_i^\pi(M)$ and $\mathsf{tu}_i^\pi(M)$ only depend on the i-prefix of π and $h \in U^{(i)}\pi$, it is sufficient to prove this lemma for $h = \pi$. If $(\pi\,M)_j = M_j$, we have $M_{1\ldots i,j} = (\pi\,M)_{1\ldots i,j}$ for each $j \in \{i+1,\ldots,e-1\}$. Since $e = \mathsf{err}_i^\pi(M)$ we find that $\mathsf{tu}_i(M)_{j-i} = \mathsf{tu}_i^\pi(M)_{j-i}$. Moreover $M_{1\ldots i,j} = \mathsf{tu}_i(M)_{j-i}$ because M is lexically ordered. Hence $(\pi\,M)_{1\ldots i,j} = \mathsf{tu}_i^\pi(M)_{j-i}$. ■

Theorem 5.5.3. *Let $M \in \mathcal{M}_n$ be lexically ordered and M_{n-1} be in canonical form. Let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1,\ldots,n-2\}$ such that $(\pi\,M)_i = M_i$ and $i+1 < \mathsf{err}_i^\pi(M) \leq n$. Write $e = \mathsf{err}_i^\pi(M)$. If $\mathsf{act}_i^\pi(M) = 1$ and*

$$M_{1\ldots i,e} < M_{1^\pi\ldots i^\pi,n} \tag{5.34}$$

then $h\,M > M$ for each $h \in U^{(i)}\,\pi$.

Proof : Since $(\pi\,M)_i$, $\mathsf{err}_i^\pi(M)$, $\mathsf{act}_i^\pi(M)$ and $M_{1^\pi\ldots i^\pi,n}$ only depend on the i-prefix of π, it is sufficient to prove this theorem for $h = \pi$. Suppose that $\pi\,M \leq M$. Hence also $(\pi\,M)_{e-1} \leq M_{e-1}$. If $(\pi\,M)_{e-1} = M_{e-1}$, then $M_{1\ldots i,j} = \mathsf{tu}_i(M)_{j-i}$ and $(\pi\,M)_{1\ldots i,j} = \mathsf{tu}_i^\pi(M)_{j-i}$ for each $j \in \{i+1,\ldots,e-1\}$, according to Lemma 5.5.2. Since $\mathsf{act}_i^\pi(M) = 1$, we therefore find that $M_{1\ldots i,e} < (\pi\,M)_{1\ldots i,k}$

for each $k \in \{e, \dots, n\}$. Hence $(\pi\, M)_e > M_e$, which clearly is a contradiction to $\pi\, M < M$.

Otherwise if $(\pi M)_{e-1} < M_{e-1}$, then there must exist an element $l \in \{i + 1, \dots, e-1\}$ such that $(\pi\, M)_{l-1} = M_{l-1}$ and $(\pi\, M)_l < M_l$. Clearly we find that $M_{1\dots i,j} = \mathsf{tu}_i(M)_{j-i}$ and $(\pi\, M)_{1\dots i,j} = \mathsf{tu}_i^{\pi}(M)_{j-i}$ for each $j \in \{i+1, \dots, l-1\}$, according to Lemma 5.5.2. Since $l < e$ and (5.34), we therefore find for each such j that $j^{\pi} \neq n$. We distinguish between the following cases:

- If $l^{\pi} = n$, then $(\pi\, M)_l > M_l$ as $M_{1\dots i,l} \leq M_{1\dots i,e}$ and (5.34).
- If $l^{\pi} \neq n$, then we find according to Lemma 5.5.1 that $(\pi\, M)_l \geq M_l$ as $g\, M_{n-1} \geq M_{n-1}$ for each $g \in \mathrm{Sym}(n-1)$.

Hence in both subcases $(\pi\, M)_l \geq M_l$ which clearly is a contradiction to $(\pi\, M)_l < M_l$. ∎

Theorem 5.5.4. *Let $M \in \mathcal{M}_n$ be lexically ordered and let $g\, M_{n-1} \geq M_{n-1}$ for each $g \in \mathrm{Sym}(n-1)$. Besides, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \dots, n-2\}$ such that $(\pi\, M)_i = M_i$ and $i+1 < \mathsf{err}_i^{\pi}(M) < n+1$. Write $e = \mathsf{err}_i^{\pi}(M)$. If $\mathsf{act}_i^{\pi}(M) = -1$ and*

$$M_{1\dots i,e} \leq M_{1^{\pi}\dots i^{\pi},n} \tag{5.35}$$

then $h\, M > M$ for each $h \in U^{(i)}\, \pi$.

Proof : Since $(\pi\, M)_i$, $\mathsf{err}_i^{\pi}(M)$, $\mathsf{act}_i^{\pi}(M)$ and $M_{1^{\pi}\dots i^{\pi},n}$ only depend on the i-prefix of π, it is sufficient to prove this theorem for $h = \pi$. Suppose that $\pi\, M \leq M$. Hence also $(\pi\, M)_{e-1} \leq M_{e-1}$. If $(\pi\, M)_{e-1} = M_{e-1}$, then $M_{1\dots i,j} = \mathsf{tu}_i(M)_{j-i}$ and $(\pi\, M)_{1\dots i,j} = \mathsf{tu}_i^{\pi}(M)_{j-i}$ for each $j \in \{i+1, \dots, e-1\}$, according to Lemma 5.5.2. Since $\mathsf{act}_i^{\pi}(M) = -1$ and (5.35), we therefore find that $\mathsf{tu}_i^{\pi}(M)_{e-i} < M_{1^{\pi}\dots i^{\pi},n}$. Hence there exists an element $k \in \{e, \dots, n\}$ such that $k^{\pi} \neq n$ and $M_{1\dots i,e} > (\pi\, M)_{1\dots i,k}$. Hence we find that

$$(\sigma'\, M)_e < M_e. \tag{5.36}$$

where $\sigma = (e\; k^{\pi^{-1}})\, \pi$. However, as $g\, M_{n-1} \geq M_{n-1}$ for each $g \in \mathrm{Sym}(n-1)$, we find according to Lemma 5.5.1 that $(\sigma M)_e \geq M_e$, which is a contradiction to (5.36).

Otherwise, if $(\pi M)_{e-1} < M_{e-1}$, then there must exist an element $l \in \{i + 1, \dots, e-1\}$ such that $(\pi\, M)_{l-1} = M_{l-1}$ and $(\pi\, M)_l < M_l$. Clearly we find that

$M_{1\ldots i,j} = \mathsf{tu}_i(M)_{j-i}$ and $(\pi\, M)_{1\ldots i,j} = \mathsf{tu}_i^\pi(M)_{j-i}$ for each $j \in \{i+1,\ldots,l-1\}$, according to Lemma 5.5.2. Since $l < e$, $\mathsf{act}_i^\pi(M) = -1$ and (5.35), we therefore find for each such j that $j^\pi \neq n$. Consider the following subcases:

1. If $l^\pi = n$, then $(\pi\, M)_l \geq M_l$ as $M_{1\ldots i,l} \leq M_{1\ldots i,e}$ and (5.35).
2. Otherwise; we find according to Lemma 5.5.1 that $(\pi\, M)_l \geq M_l$, after all $g\, M_{n-1} \geq M_{n-1}$ for each $g \in \mathrm{Sym}(n-1)$.

Hence in both subcases $(\pi\, M)_l \geq M_l$ which is a contradiction to $(\pi\, M)_l < M_l$. ■

Theorems 5.5.3 and 5.5.4 play a central role in the improvements Algorithm 5.8 makes on Algorithm 5.5. Recall that the function **rightCoset(**π, i, s**)** checks whether a permutation $h \in U^{(i-1)}\,\pi$ can be found such that $h\ M < M$. If $\mathsf{err}_{i-1}^\pi(M) > i+1$, then this function successively considers each right coset $U^{(i)}\,(i\ \ j^{\pi^{-1}})\,\pi$ with $j \in \mathsf{lex}_{i-1}^\pi(M)_{[1]}$. Write $e_{i-1} = \mathsf{err}_{i-1}^\pi(M)$. Whenever $l^\pi \neq n$ for each $l \in \{1,\ldots,i-1\}$, Algorithm 5.8 now invokes the function **rightCosetLast(**π, i, s**)** instead (❷ in Algorithm 5.8). If $\mathsf{err}_{i-1}^\pi(M) > i+1$, then the function **checkLast**(i) additionally checks whether either one of the following conditions holds (❶ in Algorithm 5.8):

$$\mathsf{act}_{i-1}^\pi(M) = 1 \quad \text{and} \quad M_{1\ldots i-1,e_{i-1}} < M_{1^\pi\ldots(i-1)^\pi,n} \tag{5.37}$$

$$\mathsf{act}_{i-1}^\pi(M) = -1 \quad \text{and} \quad M_{1\ldots i-1,e_{i-1}} \leq M_{1^\pi\ldots(i-1)^\pi,n}. \tag{5.38}$$

Recall that $g\, M_{n-1} \geq M_{n-1}$ for each $g \in \mathrm{Sym}(n-1)$. If (5.37) holds, then we find that $h\, M > M$ for each $h \in U^{(i-1)}\,\pi$, according to Theorem 5.5.3. Otherwise, if (5.38) holds, then we find that $h\, M > M$ for each $h \in U^{(i-1)}\,\pi$, according to Theorem 5.5.4. Hence in both cases no permutation of $U^{(i-1)}\,\pi$ could ever lead to a counterexample of the minimality of M.

Checking conditions (5.37) and (5.38) requires the systematic comparison of the matrix entries of $M_{1\ldots i-1,e_{i-1}}$ and $M_{1^\pi\ldots(i-1)^\pi,n}$. A more efficient way to check these conditions is in in terms of the refinement data structure. During the canonicity test an array last is used to keep track of certain nodes in the refinement tree. More precisely, let $\pi \in U^{(s-1)} - U^{(s)}$ with $s \in \{1,\ldots,n-1\}$. Then before we actually traverse $U^{(s-1)} - U^{(s)}$, we store the last node of $\mathsf{lex}[s-1]$ into $\mathsf{last}[s-1]$. Recall that the values min and max of this node, uniquely determine the last cell of $\mathsf{lex}_{s-1}(M)$. This particular node is stored because for each $h \in U^{(s-1)} - U^{(s)}$, among which π, we have

$$M_{1\dots s-1,q} < M_{1^h\dots(s-1)^h,n} = M_{1\dots s-1,p} \tag{5.39}$$

where $p \in \{\mathsf{last}[s-1].\mathsf{min},\dots,n\}$ and $q \in \{s,\dots,\mathsf{last}[s-1].\mathsf{min}-1\}$. While traversing $U^{(s-1)} - U^{(s)}$, the canonicity algorithm subsequently encounters the right cosets

$$U^{(s)}\,\pi,\ U^{(s+1)}\,\pi,\ U^{(s+2)}\,\pi\ \dots,\ U^{(i-1)}\,\pi,\ U^{(i)}\,\pi$$

descending each time one level further in the recursion tree. When the canonicity algorithm is about to traverse a right coset $U^{(l)}\,\pi$ with $l \in \{s,\dots,i\}$, then we adjust $\mathsf{last}[l]$ to contain either the node $\mathsf{last}[l-1].\mathsf{ref}[M_{l^\pi,n}]$ if $\mathsf{last}[l-1]$ is a non-dummy node or the node $\mathsf{last}[l-1]$ otherwise. To put it differently, at the same time as we descend on level in the recursion tree, we descend one level in the refinement tree.

Moreover, if the node $\mathsf{last}[l]$ itself is a non-dummy node, then for each $h \in U^{(l)}\,\pi$ we have

$$M_{1\dots l,p} < M_{1^h\dots l^h,n} = M_{1\dots l,q} < M_{1\dots l,r} \tag{5.40}$$

where $p \in \{l+1,\dots,\mathsf{last}[l].\mathsf{min}-1\}$, $q \in \{\mathsf{last}[l].\mathsf{min},\dots,\mathsf{last}[l].\mathsf{max}\}$ and $r \in \{\mathsf{last}[l].\mathsf{max}+1,\dots,n\}$. Otherwise, if the node $\mathsf{last}[l]$ itself is a dummy node, then for each $h \in U^{(l)}\,\pi$ we have

$$M_{1\dots l,p} < M_{1^h\dots l^h,n} < M_{1\dots l,r} \tag{5.41}$$

where $p \in \{l+1,\dots,\mathsf{last}[l].\mathsf{max}\}$ and $r \in \{\mathsf{last}[l].\mathsf{min},\dots,n\}$. Before we actually traverse the right coset $U^{(l)}\,\pi$, we can now use the information stored in $\mathsf{last}[l]$ to check conditions (5.37) and (5.37) for $i = l+1$. More precisely, by (5.39), (5.40) and (5.41), condition (5.37) translates to checking whether $\mathsf{act}_l^\pi(M) = 1$ and $\mathsf{err}_l^\pi(M) < \mathsf{last}[l].\mathsf{min}$, whereas condition (5.38) translates to checking whether $\mathsf{act}_l^\pi(M) = -1$ and $\mathsf{err}_l^\pi(M) \leq \mathsf{last}[l].\mathsf{max}$. Note that this translation is independent of the node $\mathsf{last}[l]$ being a non-dummy node or not.

Finally, the question remains whether we still obtain, as in Algorithm 5.5, a strong generating set $S = S^{(0)}$ for the automorphism group $\operatorname{Aut} M$ with base $[1,\dots,n]$ whenever M is minimal. Let $\pi \in (\operatorname{Aut} M)^{(s)} - (\operatorname{Aut} M)^{(s+1)}$, then

$$\mathsf{err}_p^\pi(M) = n+1 \quad \text{and} \quad \mathsf{act}_p^\pi(M) = 0$$

Algorithm 5.8 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function isCanonical($M \in \mathcal{M}_n$) : boolean
for $i \leftarrow n-1, \dots 1$ do
    if diffStab($i$) $< 0$ then
        return false
return true

function diffStab($i$ : int) : int
for all $j \in (\mathsf{lex}_{i-1}(M)_{[1]} \setminus \{i\})$ do
    if $j = \min(j^{\langle S^{(i-1)}_{\mathrm{Aut}M} \rangle})$ then
        refine($(i\ j)$, $i$)
        if $j \neq n$ then                                        ❷
            $d \leftarrow$ rightCosetLast($(i\ j)$, $i+1$, $i-1$)
        else
            $d \leftarrow$ rightCoset($(i\ j)$, $i+1$, $i-1$)
        if $d < 0$ then
            return $d$
return 1

function rightCoset($\pi \in U^{(s)} - U^{(s+1)}$, $i, s$ : int) : int
if checkErrorLevel($\pi$, $i$, $s$) then
    return $\mathsf{act}^{\pi}_{i-1}(M)$
else
    for all $j \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$ do
        $\pi' \leftarrow (i\ j^{\pi^{-1}})\ \pi$
        refine($\pi'$, $i$)
        $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
        if $d < 0 \vee d = 0$ then
            return $d$
    return 1
```

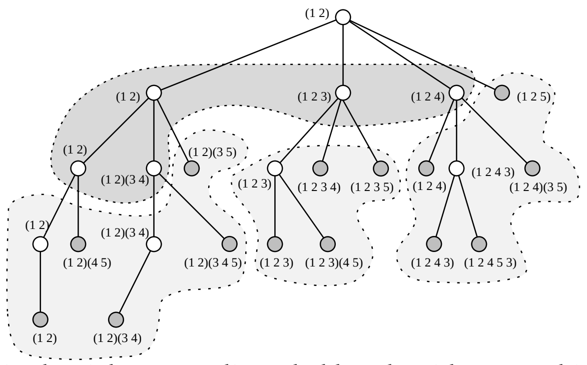
Algorithm 5.8 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function rightCosetLast($\pi \in U^{(s)} - U^{(s+1)}$, $i, s$ : int) : int
 1: if checkErrorLevel($\pi$, $i$, $s$) then
 2:    return $\mathsf{act}^{\pi}_{i-1}(M)$
 3: else if checkLast($i$) then                                    ❶
 4:    return 1
 5: else
 6:    for all $j \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$ do
 7:       $\pi' \leftarrow (i\ j^{\pi^{-1}})\ \pi$
 8:       refine($\pi'$, $i$)
 9:       if $j \neq n$ then                                       ❷
10:          $d \leftarrow$ rightCosetLast($\pi'$, $i+1$, $s$)
11:       else
12:          $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
13:       if $d < 0 \vee d = 0$ then
14:          return $d$
15:    return 1

function checkErrorLevel($\pi \in U^{(s)} - U^{(s+1)}$, $i, s$ : int) : boolean
 1: if $\mathsf{err}^{\pi}_{i-1}(M) \leq i+1$ then
 2:    if $\mathsf{err}^{\pi}_{i-1}(M) = n+1$ then
 3:       $\bar{S}^{(s)}_{\mathrm{Aut}\,M} \leftarrow \bar{S}^{(s)}_{\mathrm{Aut}\,M} \cup \pi$
 4:    return true
 5: else
 6:    return false

function checkLast($\pi \in \mathrm{Sym}(n)$, $i$ : int) : boolean
 1: $e_{i-1} \leftarrow \mathsf{err}^{\pi}_{i-1}(M)$
 2: return $(\mathsf{act}^{\pi}_{i-1}(M) = 1 \wedge M_{1\ldots i-1,e_{i-1}} < M_{1^{\pi}\ldots(i-1)^{\pi},n}) \vee$
             $(\mathsf{act}^{\pi}_{i-1}(M) = -1 \wedge M_{1\ldots i-1,e_{i-1}} \leq M_{1^{\pi}\ldots(i-1)^{\pi},n})$
```

for each $p \in \{s+1, \dots, n-1\}$. Therefore the additional constraints incorporated in Algorithm 5.8 will never discard a right coset which contains an automorphism of M. Hence when M is minimal, Algorithm 5.8 still obtains a strong generating set $S = S^{(0)}$ for the automorphism group $\operatorname{Aut} M$ with base $[1, \dots, n]$.

Example 5.25. Let $\sigma = (1\ 2)$. Consider the lexically ordered matrix $M \in \mathcal{M}_5$ below. Note that M_4 is minimal. The recursion tree shown corresponds to the traversal of $U^{(1)}\,\sigma$ by Algorithm 5.4.

$$M = \begin{pmatrix} 0 \,| & 1 & 1 & 1 & 1 \\ 1 & 0 \,| & 1 & 1 \,| & 2 \\ 1 & 1 & 0 \,| & 2 \,| & 1 \\ 1 & 1 & 2 & 0 \,| & 2 \\ 1 & 2 & 1 & 2 & 0 \end{pmatrix}$$



The light shaded areas contain the right cosets discarded by Algorithm 5.5 while the dark shaded area contains the right cosets which are now also discarded by Algorithm 5.8. Note that $\mathsf{lex}_1^{\sigma}(M) = (\{1,3,4\},\{5\})$, $\mathsf{lex}_1(M) = (\{2,3,4,5\})$ and

$$\begin{aligned} \mathsf{tu}_1(M) &= \{(1),(1),(1),(\mathbf{1})\} \\ \mathsf{tu}_1^{\sigma}(M) &= \{(1),(1),(1),(\mathbf{2})\}. \end{aligned}$$

Hence clearly $e = \mathsf{err}_1^{\sigma}(M) = 5$, $\mathsf{act}_1^{\sigma}(M) = 1$. Therefore we find that only the right cosets $U^{(2)}\,(1\,2)$, $U^{(2)}\,(1\,2\,3)$ and $U^{(2)}\,(1\,2\,4)$ need to be considered as $\mathsf{lex}_1^{\sigma}(M)_{[1]} = \{1,3,4\}$. However, because $M_{1,e} < M_{1^{\sigma},5}$ and M_4 is minimal, we find that $h\,M > M$ for each $h \in U^{(1)}\,\sigma$, according to Theorem 5.5.3. Hence we may additionally discard the right cosets $U^{(2)}\,(1\,2)$, $U^{(2)}\,(1\,2\,3)$ and $U^{(2)}\,(1\,2\,4)$ from traversal. •

5.5.2 Automorphism group of the leading principal submatrices

The successive application of our canonicity algorithm and the fact that each submatrix M_i with $i \in \{2, \ldots, n-1\}$ is minimal, guarantees us that a *strong generating set* $S^{(0)}_{\mathrm{Aut}\, M_i}$ for the *automorphism group* of each such submatrix M_i has been determined. Below we shall outline three different strategies which exploit these strong generating sets to further improve our canonicity algorithm.

The *first strategy* uses these strong generating sets to *guess* some of the automorphisms of M. More precisely, assume that before we traverse $\mathrm{Sym}(n)$, we already know a subgroup G of $\mathrm{Aut}\, M$. Let $s \in \{n-2, \ldots, 0\}$, then before we traverse $U^{(s)} - U^{(s+1)}$, we could add the elements of the stabilizer $G^{(s)}$ to $(\mathrm{Aut}\, M)^{(s)}$ which we are constructing during traversal. Generally, we will know a strong generating set S_G of G. Hence the addition of the elements of $G^{(s)}$ to $(\mathrm{Aut}\, M)^{(s)}$ amounts to including the elements of $\bar{S}^{(s)}_G$ to $\bar{S}^{(s)}_{\mathrm{Aut}\, M}$. Note that at the same time we should update the corresponding orbit partition accordingly.

v	k	λ	μ	type I/all	type II/all	v	k	λ	μ	type I/all	type II/all
16	5	0	2	0.547	0.333	29	14	6	7	0.647	0.148
17	8	3	4	0.490	0.333	36	10	4	2	0.830	0.133
25	8	3	2	0.756	0.182	36	14	4	6	0.533	0.223
25	12	5	6	0.505	0.224	36	14	7	4	0.800	0.167
26	10	3	4	0.440	0.358	40	12	2	4	0.962	0.019
27	10	1	5	0.619	0.212	45	16	8	4	0.821	0.149
28	12	6	4	0.684	0.224	50	7	0	1	0.712	0.137

Tabel 5.5: Ratio of automorphisms of type I and type II obtained during the orderly generation of strongly regular graphs.

We say that a permutation $\pi \in \mathrm{Aut}\, M$ is an automorphism of *type I* if it stabilizes the element n. We say that π is an automorphism of *type II* if it satisfies $(n-1)^\pi = n$ and $n^\pi = n-1$. Table 5.5 represents empirical data obtained from the orderly generation of strongly regular graphs. The ratio of automorphisms of type I and type II encountered by canonicity Algorithm 5.5 is listed. The large ratio of automorphisms of type I, and to a lesser extent of type II, suggests that we can use the strong generating sets $S^{(0)}_{\mathrm{Aut}\, M_{n-1}}$ and $S^{(0)}_{\mathrm{Aut}\, M_{n-2}}$ to easily acquire some of the generators of $S^{(0)}_{\mathrm{Aut}\,M}$. Algorithm 5.9 illustrates the use of these strong generating sets. For future notational simplicity we assume that $\bar{S}^{(b)}_{\mathrm{Aut}\,M_a}$ denotes the empty set whenever $a < 2$ or $b < 0$. Prior to the traversal of $U^{(s)} - U^{(s+1)}$ with $s \in \{n-2, \dots, 0\}$, we apply one or both of the following techniques (❶ in Algorithm 5.9):

1. For each $\pi \in \bar{S}^{(s)}_{\mathrm{Aut}\,M_{n-1}}$ we check whether j^π is the minimal element in $(j^\pi)^{\langle S^{(s)}_{\mathrm{Aut}M}\rangle}$. If so we then check whether
$$M_{(s+1)^\pi \dots (n-1)^\pi, n} = M_{s+1 \dots n-1, n}.$$
If both tuples are equal, then extension of π to $\{1, \dots, n\}$ (with $n^\pi = n$), abbreviated by π_{typeI}, is an automorphism of M of type I. Note that π_{typeI} is added to $\bar{S}^{(s)}_{\mathrm{Aut}\,M}$ and the corresponding orbit partition should be updated accordingly (❷ in Algorithm 5.9).

2. For each $\pi \in \bar{S}^{(s)}_{\mathrm{Aut}\,M_{n-2}}$ we check whether j^π is the minimal element in $(j^\pi)^{\langle S^{(s)}_{\mathrm{Aut}M}\rangle}$. If so, we then check whether
$$M_{1^\pi \dots (n-2)^\pi, n} = M_{1 \dots n-2, n-1} \quad \text{and} \quad M_{1^\pi \dots (n-2)^\pi, n-1} = M_{1 \dots n-2, n}.$$
If both equalities hold, then the extension of π to $\{1, \dots, n\}$ with $(n-1)^\pi = n$ and $n^\pi = n-1$, abbreviated by π_{typeII}, is an automorphism of M of type II. Note that π_{typeII} is added to $\bar{S}^{(s)}_{\mathrm{Aut}\,M}$ and the corresponding orbit partition should be updated accordingly (❸ in Algorithm 5.9).

Because of the very low overhead cost involved, we apply the first refinement before the traversal of each stabilizer difference $U^{(s)} - U^{(s+1)}$. We apply the second refinement only before the traversal of stabilizer differences $U^{(s)} - U^{(s+1)}$ with relatively small s. The occurrence of automorphism of type II is less frequent and this technique involves a somewhat higher overhead cost. But still, at shallow levels we cannot ignore the possible advantage that large cosets may be discarded from traversal. Finally note that as a side effect of both refinements we may find a counterexample to the minimality of M.

The application of our canonicity algorithm during a single generation guarantees us that the matrix $M \in \mathcal{M}_n$ must be 0-lexically ordered. As a result of this we can check instantly whether or not a given transposition $(k\, k+1)$ is an automorphism of M_l with $l \in \{2, \dots, n\}$ and $k \in \{1, \dots, l-1\}$. The *second strategy* uses these transpositions to further optimize our canonicity algorithm.

Theorem 5.5.5. *Let the matrix $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1, \dots, n-2\}$ such that $(\pi\, M)_{i+1} = M_{i+1}$ and*

$$\mathsf{err}^{\pi}_{i-1}(M) = \mathsf{err}^{\pi}_{i}(M) = \mathsf{err}^{\pi}_{i+1}(M) > i+1 \tag{5.42}$$

Write $e = \mathsf{err}^{\pi}_{i-1}(M)$ and $\rho = (i\, i+1)$. If $(\rho\, M)_{e-1} = M_{e-1}$, then we have

$$\mathsf{err}^{\rho\,\pi}_{i}(M) = \mathsf{err}^{\pi}_{i}(M), \quad \mathsf{act}^{\rho\,\pi}_{i}(M) = \mathsf{act}^{\pi}_{i}(M), \quad \mathsf{piv}^{\rho\,\pi}_{i}(M) = \mathsf{piv}^{\pi}_{i}(M).$$

Moreover, if $g\, M > M$ for each $g \in U^{(i)}\, \pi$, then $h\, M > M$ for each $h \in U^{(i)}\, \rho\, \pi$.

Proof : (1) For each $x \in \{i+2, \dots, n\}$ we may write

$$\mathsf{tu}^{\pi}_{i+1}(M)_{x-(i+1)} = (\pi\, M)_{1\dots i+1, j_x} \tag{5.43}$$

with $j_x \in \{i+2, \dots, n\}$. Also note that $\mathsf{tu}_{i+1}(M)_{x-(i+1)} = M_{1\dots i+1,x}$ for each such x. Moreover, by (5.42) we have

$$\mathsf{tu}_{i+1}(M)_{k-(i+1)} = \mathsf{tu}^{\pi}_{i+1}(M)_{k-(i+1)} \tag{5.44}$$

when $k \in \{i+2, \dots, e-1\}$. Hence for such k we find that

$$(\pi\, M)_{1\dots i+1, j_k} = \mathsf{tu}^{\pi}_{i+1}(M)_{k-(i+1)} = \mathsf{tu}_{i+1}(M)_{k-(i+1)} = M_{1\dots i+1,k}$$

and in particular $(\pi\, M)_{i,j_k} = M_{i,k}$ and $(\pi\, M)_{i+1,j_k} = M_{i+1,k}$. Therefore, as $(\rho\, M)_{e-1} = M_{e-1}$ we have

$$\begin{aligned} (\pi\, M)_{i,j_k} &= M_{i,k} = M_{i^\rho,k^\rho} = M_{i+1,k} = (\pi\, M)_{i+1,j_k} \\ &= M_{(i+1)^\pi, j_k{}^\pi} = M_{(i+1)^{\rho\,\pi}, j_k{}^{\rho\,\pi}} \\ &= (\rho\, \pi\, M)_{i,j_k}. \end{aligned} \tag{5.45}$$

and similarly $(\pi\, M)_{i+1,j_k} = (\rho\, \pi\, M)_{i+1,j_k}$. Also we find for each $j,\ l \notin \{i, i+1\}$ that

$$(\pi\, M)_{j,l} = M_{j^\pi, l^\pi} = M_{j^{\rho\,\pi}, l^{\rho\,\pi}} = (\rho\, \pi\, M)_{j,l}. \tag{5.46}$$

Combining (5.46) and (5.45) we find for each $k \in \{i+2, \dots, e-1\}$ and $l \in \{e, \dots, n\}$ that

$$(\rho\, \pi\, M)_{1\dots i+1, j_k} = (\pi\, M)_{1\dots i+1, j_k} \text{ and } (\rho\, \pi\, M)_{1\dots i-1, j_l} = (\pi\, M)_{1\dots i-1, j_l}. \tag{5.47}$$

Using (5.42) we find that

$$\begin{aligned}\mathsf{piv}^{\pi}_{i-1}(M) = \mathsf{piv}^{\pi}_{i}(M) = \mathsf{piv}^{\pi}_{i+1}(M)\\ \mathsf{act}^{\pi}_{i-1}(M) = \mathsf{act}^{\pi}_{i}(M) = \mathsf{act}^{\pi}_{i+1}(M)\end{aligned} \tag{5.48}$$

according to Theorem 5.4.17 and Corollary 5.4.1. Write $p = \mathsf{piv}^{\pi}_{i+1}(M)$. Note that $p \leq i-1$. We distinguish between the following cases:

- If $\mathsf{act}^{\pi}_{i+1}(M) = 1$, then $e \leq n$ and $p \neq 0$. Hence for each $l \in \{e, \dots, n\}$ we have that $(\pi M)_{1\dots p, j_e} \leq (\pi M)_{1\dots p, j_l}$ and

  $$M_{1\dots p-1,e} = (\pi M)_{1\dots p-1, j_e} \quad M_{p,e} < (\pi M)_{p, j_e} \tag{5.49}$$

  By (5.44), (5.47) and (5.49), we therefore find that

  $$\begin{aligned}\mathsf{err}^{\rho\,\pi}_{i+1}(M) &= \mathsf{err}^{\pi}_{i+1}(M)\\ \mathsf{act}^{\rho\,\pi}_{i+1}(M) &= \mathsf{act}^{\pi}_{i+1}(M)\\ \mathsf{piv}^{\rho\,\pi}_{i+1}(M) &= \mathsf{piv}^{\pi}_{i+1}(M).\end{aligned} \tag{5.50}$$

- If $\mathsf{act}^{\pi}_{i+1}(M) = -1$, then $e \leq n$ and $p \neq 0$. Hence for each $l \in \{e, \dots, n\}$ we have that $(\pi M)_{1\dots p, j_e} \leq (\pi M)_{1\dots p, j_l}$ and

  $$M_{1\dots p-1,e} = (\pi M)_{1\dots p-1, j_e} \quad M_{p,e} > (\pi M)_{p, j_e} \tag{5.51}$$

  By (5.44), (5.47) and (5.51), we find again that (5.50) holds.
- If $\mathsf{act}^{\pi}_{i+1}(M) = 0$, then $e = n+1$ and $p = 0$. By (5.44) and (5.47) we find again that (5.50) holds.

Since $\rho\pi \in U^{(i-1)}\pi$ we have $\mathsf{err}^{\pi}_{i-1}(M) = \mathsf{err}^{\rho\,\pi}_{i-1}(M)$. By (5.50) we find according to Theorem 5.4.17 and Corollary 5.4.1 that $\mathsf{err}^{\rho\,\pi}_{i}(M) = \mathsf{err}^{\pi}_{i}(M)$, $\mathsf{act}^{\rho\,\pi}_{i}(M) = \mathsf{act}^{\pi}_{i}(M)$, $\mathsf{piv}^{\rho\,\pi}_{i}(M) = \mathsf{piv}^{\pi}_{i}(M)$ which proves the first part of the theorem.

(2) Since $(\pi M)_{i+1} = M_{i+1}$ and $\mathsf{err}^{\pi}_{i}(M) = \mathsf{err}^{\pi}_{i+1}(M)$, we have

$$\mathsf{tu}_i(M)_1 = \mathsf{tu}^{\pi}_i(M)_1 = (\pi M)_{1\dots i, i+1} \tag{5.52}$$

and for each $x \in \{i+2, \dots, e-1\}$ we have

$$\mathsf{tu}_i(M)_{x-i} = \mathsf{tu}^{\pi}_i(M)_{x-i} = (\pi M)_{1\dots i, j_x} \tag{5.53}$$

where j_x is the same as in (5.43). Moreover, since $(\rho\,\pi\,M)_{i+1} = M_{i+1}$ and $\mathsf{err}^{\rho\,\pi}_{i+1}(M) = \mathsf{err}^{\rho\,\pi}_{i}(M) = \mathsf{err}^{\pi}_{i}(M)$, we have

$$\mathsf{tu}_i(M)_1 = \mathsf{tu}^{\rho\,\pi}_i(M)_1 = (\rho\,\pi\,M)_{1\ldots i,i+1} \tag{5.54}$$

and using (5.47) and (5.50) we find that

$$\mathsf{tu}_i(M)_{x-i} = \mathsf{tu}^{\rho\,\pi}_i(M)_{x-i} = (\rho\,\pi\,M)_{1\ldots i,j_x} \tag{5.55}$$

for each such x and j_x. For each $h \in U^{(i)}\,\rho\,\pi$ there exist a unique $g \in U^{(i)}\,\pi$ such that $h = g\;(i^\pi\;(i+1)^\pi)$. Because $g\,M > M$, there exists an $s \in \{i+1,\ldots,n\}$ such that $(g\,M)_{s-1} = M_{s-1}$ and $(g\,M)_s > M_s$. Hence, as $g \in U^{(i)}\,\pi$ and $\mathsf{err}^{\pi}_i(M) = e$ we find that $s \le e$. Indeed, if $s > e$ then $(g\,M)_e = M_e$ and then for all $j \in \{i+1,\ldots,e\}$

$$\mathsf{tu}_i(M)_{j-i} = M_{1\ldots i,j} = (g\,M)_{1\ldots i,j} = \mathsf{tu}^{\pi}_i(M)_{j-i} \tag{5.56}$$

which contradicts $\mathsf{err}^{\pi}_i(M) = e$. Hence for $j \in \{i+1,\ldots,s-1\}$ we find that

$$\begin{aligned} j^g \in V &= \{(i+1)^\pi\} \cup \{{j_{i+2}}^\pi,\ldots,{j_{e-1}}^\pi\} \cup X \\ j^h \in W &= \{i^\pi\} \cup \{{j_{i+2}}^\pi,\ldots,{j_{e-1}}^\pi\} \cup X \end{aligned} \tag{5.57}$$

where ${j_x}^\pi \in X$ if and only if $x \ge e$ and $\mathsf{tu}^{\pi}_i(M)_{x-i} = \mathsf{tu}^{\pi}_i(M)_{e-1-i}$.

Write $y = (i+1)^{\pi g^{-1}}$. Hence we have

$$i^g = i^\pi, \quad y^g = (i+1)^\pi, \quad i^h = (i+1)^\pi, \quad y^h = i^\pi \quad \text{and} \quad p^g = p^h \tag{5.58}$$

when $p \notin \{i,l\}$. Moreover, the following relations hold:

- If k, $m \notin \{i,l\}$, then using (5.58) we have
$$(g\,M)_{k,m} = (h\,M)_{k,m}. \tag{5.59}$$
- If $k < i$ and $m \in \{i,y\}$, then using (5.58) we have

$$\begin{aligned} (g\,M)_{k,i} &= M_{k^g,i^g} = M_{k^\pi,i^\pi} = (\pi\,M)_{k,i} \\ &= (\rho\,\pi\,M)_{k,i} = M_{k^{\rho\,\pi},i^{\rho\,\pi}} = M_{k^\pi,(i+1)^\pi} \\ &= M_{k^h,i^h} = (h\,M)_{k,i} \qquad (5.60) \\ (g\,M)_{k,y} &= M_{k^g,y^g} = M_{k^\pi,(i+1)^\pi} = (\pi\,M)_{k,i+1} \\ &= (\rho\,\pi\,M)_{k,i+1} = M_{k^{\rho\,\pi},(i+1)^{\rho\,\pi}} = M_{k^\pi,i^\pi} \\ &= M_{k^h,y^h} = (h\,M)_{k,y} \qquad (5.61) \end{aligned}$$

because $g, h \in U^{(i-1)}\,\pi$ and $(\rho\,\pi\,M)_{i+1} = (\pi\,M)_{i+1}$,

- If $k^g \in V \setminus \{(i+1)^\pi\}$, among which k^g with $k \in \{i+2, \ldots, s-1\} \setminus \{y\}$, and $m \in \{i, y\}$, then using (5.45) and (5.58) we have

$$\begin{aligned}
(g\,M)_{k,i} &= M_{k^g,i^\pi} = M_{j_x^\pi,i^\pi} = (\pi\,M)_{j_x,i} \\
&= (\rho\,\pi\,M)_{j_x,i} = M_{j_x{}^{\rho\pi},i^{\rho\pi}} = M_{j_x{}^{\pi},(i+1)^\pi} \\
&= M_{k^\pi,(i+1)^\pi} = (h\,M)_{k,i} \qquad (5.62) \\
(g\,M)_{k,y} &= M_{k^g,(i+1)^\pi} = M_{j_x^\pi,(i+1)^\pi} = (\pi\,M)_{j_x,i+1} \\
&= (\rho\,\pi\,M)_{j_x,i+1} = M_{j_x{}^{\rho\pi},(i+1)^{\rho\pi}} = M_{j_x{}^{\pi},i^\pi} \\
&= M_{k^\pi,i^\pi} = (h\,M)_{k,y} \qquad (5.63)
\end{aligned}$$

where $x \in \{i+2, \ldots, e-1\}$.

- Finally we have $(g\,M)_{i,i} = (h\,M)_{l,l}$ and

$$(g\,M)_{i,y} = M_{i^\pi,(i+1)^\pi} = M_{(i+1)^\pi,i^\pi} = (h\,M)_{i,y}. \qquad (5.64)$$

Hence for each $k, m \in \{1, \ldots, s-1\}$ we have $(g\,M)_{k,m} = (h\,M)_{k,m}$ and therefore

$$(h\,M)_{s-1} = (g\,M)_{s-1} = M_{s-1}. \qquad (5.65)$$

Since $(g\,M)_s > M_s$ we have $(g\,M)_{1\ldots s-1,s} > M_{1\ldots s-1,s}$. If $s^g \in V$, then by (5.59–5.64) we have

$$M_{1\ldots s-1,s} < (g\,M)_{1\ldots s-1,s} = (h\,M)_{1\ldots s-1,s}. \qquad (5.66)$$

Otherwise, if $s^g \notin V$, then we distinguish between the following cases:

- Let $\mathrm{act}_i^\pi(M) = 1$ and let $s^g = j_x{}^\pi$ with $x \in \{e, \ldots, n\}$, then using (5.59) we find that

$$M_{1\ldots p,s} \leq M_{1\ldots p,e} < (\pi\,M)_{1\ldots p,j_x} = (g\,M)_{1\ldots p,s} = (h\,M)_{1\ldots p,s}. \qquad (5.67)$$

- Let $\mathrm{act}_i^\pi(M) = -1$. Note that in this particular case $s \neq e$; otherwise $\sigma\,M < M$ where $\sigma = g\,(e^g\ j_e{}^\pi) \in U^{(i)}\,\pi$. Let $s^g = j_x{}^\pi$ with $x \in \{e, \ldots, n\}$, then using (5.59) we find that

$$M_{1\ldots p,s} \leq M_{1\ldots p,e-1} < (\pi\,M)_{1\ldots p,j_x} = (g\,M)_{1\ldots p,s} = (h\,M)_{1\ldots p,s}. \qquad (5.68)$$

- Let $\mathsf{act}_i^{\pi}(M) = 0$. Note that in this particular case $s \neq e$, because $e = n+1$. Clearly we have $j^g \in V$.

Hence using (5.65) and (5.66–5.68), we find that $(h\,M)_s > M_s$ which proves the second part of the theorem. ■

Corollary 5.5.1. *Let the matrix $M \in \mathcal{M}_n$ be lexically ordered, let $\pi \in \mathrm{Sym}(n)$ and $i \in \{1,\ldots,n-2\}$, $k \in \{1,\ldots,n-i-1\}$ such that $(\pi\,M)_{i+k} = M_{i+k}$ and*

$$\mathsf{err}_{i-1}^{\pi}(M) = \mathsf{err}_i^{\pi}(M) = \ldots = \mathsf{err}_{i+k}^{\pi}(M) > i+k \tag{5.69}$$

Write $e = \mathsf{err}_{i-1}^{\pi}(M)$ and $\rho = (i\;i+k)$. If $(\rho\,M)_{e-1} = M_{e-1}$, then we have

$$\mathsf{err}_i^{\rho\,\pi}(M) = \mathsf{err}_i^{\pi}(M), \quad \mathsf{act}_i^{\rho\,\pi}(M) = \mathsf{act}_i^{\pi}(M), \quad \mathsf{piv}_i^{\rho\,\pi}(M) = \mathsf{piv}_i^{\pi}(M).$$

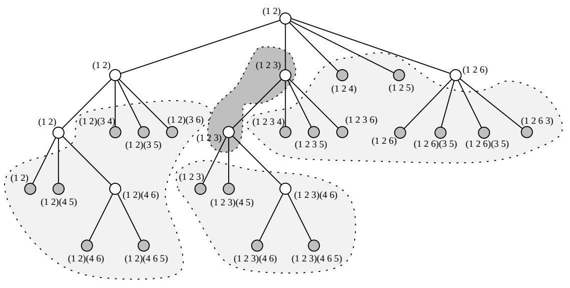
Moreover, if $g\,M > M$ for each $g \in U^{(i)}\,\pi$, then $h\,M > M$ for each $h \in U^{(i)}\,\rho\,\pi$. ◇

Corollary 5.5.1 plays a central role in some of the improvements Algorithm 5.9 makes on Algorithm 5.8. Recall that the functions **rightCoset**(π, i ,s) and **rightCosetLast**(π, i ,s) check whether a permutation $h \in U^{(i-1)}\,\pi \subset U^{(s)} - U^{(s+1)}$ can be found such that $h\,M < M$. If still necessary, these functions successively consider each right coset $U^{(i)}\,\pi'$ with $\pi' = (i\;j^{\pi^{-1}})$ and $j \in \mathsf{lex}_{i-1}^{\pi}(M)_{[1]}$. If such a right coset $U^{(i)}\,\pi'$ turns out not to contain any counterexample to the minimality of M nor any automorphism of M, Algorithm 5.9 now systematically invokes the method **subTranspose**(π', i ,s) (❺ in Algorithm 5.9).

For each level $l \in \{1,\ldots,n-1\}$ in the recursion, Algorithm 5.9 maintains an additional set C_l of marks. Write $e = \mathsf{err}_i^{\pi'}(M)$, $p = \mathsf{piv}_i^{\pi'}(M)$ and $q = \max(s+1,p)+1$. For each $j \in \{i-1,\ldots,q\}$, the method call **subTranspose**(π', i ,s) adds the mark $i^{\pi'}$ to the set C_j whenever $(j\;i) \in \mathrm{Aut}\,M_{e-1}$. Once we have considered the right coset $U^{(i-1)}\,\pi$ entirely and have not found a counterexample to the minimality of M, Algorithm 5.9 now uses the stored marks to discard additional right cosets. More precisely, when in the remainder of the traversal of $U^{(q-1)}\,\pi$ we are about to consider a right coset $U^{(j)}\,\tau$ with $j \in \{q,\ldots,i-1\}$ such that the mark $j^{\tau} \in C_j$, we can discard this right coset $U^{(j)}\,\tau$ entirely (❹ in Algorithm 5.9). Indeed $h\,M > M$ for each $h \in U^{(j)}\,\tau$, according to Corollary 5.5.1.

Example 5.26. Let $\sigma = (1\ 2)$. Consider the lexically ordered matrix $M \in \mathcal{M}_6$ below. The recursion tree shown corresponds to the traversal of $U^{(1)}\,\sigma$ by Algorithm 5.4.

$$M = \left(\begin{array}{c|c|cccc|c} 0 & 1 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 2 & 1 \\ 1 & 1 & 0 & 2 & 2 & 2 \\ 1 & 2 & 2 & 0 & 1 & 1 \\ 1 & 2 & 2 & 1 & 0 & 2 \\ 2 & 1 & 2 & 1 & 2 & 0 \end{array}\right)$$



The light shaded areas contain the right cosets discarded by Algorithm 5.5 while the dark shaded area contains the right cosets which are now also discarded by Algorithm 5.9. Note that $\mathsf{lex}_1^{\sigma}(M) = (\{1,3,6\},\{4,5\})$, $\mathsf{lex}_1(M) = (\{2,3,4,5\},\{6\})$ and

$$\begin{aligned} \mathsf{tu}_1(M) &= \{(1),(1),(1),(\mathbf{1}),(2)\} \\ \mathsf{tu}_1^{\sigma}(M) &= \{(1),(1),(1),(\mathbf{2}),(2)\}. \end{aligned}$$

Hence $\mathsf{err}_1^{\sigma}(M) = 5$. Therefore we find that only the right cosets $U^{(2)}\,(1\,2)$, $U^{(2)}\,(1\,2\,3)$ and $U^{(2)}\,(1\,2\,6)$ need to be considered as $\mathsf{lex}_1^{\sigma}(M)_{[1]} = \{1,3,6\}$.

While traversing the right coset $U^{(2)}\,(1\,2)$ we find that $g\,M > M$ for each $g \in U^{(2)}\,(1\,2)$. Moreover $\mathsf{err}_2^{\sigma}(M) = \mathsf{err}_3^{\sigma}(M) = 5$. Since $(2\ 3) \in (\operatorname{Aut} M_4)^{(1)}$, we find that $h\,M > M$ for each $h \in U^{(2)}\,(1\,2\,3)$, according to Corollary 5.5.1. Hence we may additionally discard the right coset $U^{(2)}\,(1\,2\,3)$. •

In order to outline the *third strategy* we introduce the following theorems:

Theorem 5.5.6. *Let $M \in \mathcal{M}_n$ be lexically ordered. Let $i \in \{2,\dots,n-2\}$ and $\pi \in \operatorname{Sym}(n)$ such that*

$$e = \mathsf{err}_{i-1}^{\pi}(M) > i, \quad \mathsf{act}_{i-1}^{\pi}(M) = 1, \quad \textit{and} \quad (\pi\,M)_{e-1} = M_{e-1}.$$

Let $\rho \in U^{(i-1)}$ such that $(\rho\,M)_{e-1} = M_{e-1}$ and

$$\{i^{\rho},\dots,(e-1)^{\rho}\} = \{i,\dots,e-1\} \quad \textit{and} \quad \{e^{\rho},\dots,n^{\rho}\} = \{e,\dots,n\}.$$

If $g\,M > M$ for each $g \in U^{(i)}\,\pi$, then $h\,M > M$ for each $h \in U^{(i)}\,\rho\,\pi$.

Proof : Let $x,\ y \in \{1,\ldots,e-1\}$ and write $k = x^{\rho}$, $l = y^{\rho}$ then $k, l \in \{1,\ldots,e-1\}$. Since $(\pi\,M)_{e-1} = (\rho\,M)_{e-1} = M_{e-1}$ we have

$$M_{x^{\pi},y^{\pi}} = M_{x,y} = M_{x^{\rho},y^{\rho}} = M_{k,l} = M_{k^{\pi},l^{\pi}} = M_{x^{\rho\pi},y^{\rho\pi}}. \tag{5.70}$$

Hence we find that also $(\rho\,\pi\,M)_{e-1} = M_{e-1}$.

For each $x \in \{i,\ldots,n\}$ we may write

$$\mathsf{tu}^{\pi}_{i-1}(M)_{x-(i-1)} = (\pi\,M)_{1\ldots i-1,j_x} \tag{5.71}$$

with $j_x \in \{i,\ldots,n\}$. Also note that $\mathsf{tu}_{i-1}(M)_{x-(i-1)} = M_{1\ldots i-1,x}$ for each such x. Moreover, because $e = \mathsf{err}^{\pi}_{i-1}(M)$ we have

$$\mathsf{tu}_{i-1}(M)_{k-(i-1)} = \mathsf{tu}^{\pi}_{i-1}(M)_{k-(i-1)} \tag{5.72}$$

when $k \in \{i,\ldots,e-1\}$. Hence for such k we find that

$$(\pi\,M)_{1\ldots i-1,j_k} = \mathsf{tu}^{\pi}_{i-1}(M)_{k-(i-1)} = \mathsf{tu}_{i-1}(M)_{k-(i-1)} = M_{1\ldots i-1,k}$$

Therefore, as $\mathsf{act}^{\pi}_{i-1}(M) = 1$ and $(\pi\,M)_{1\ldots i-1,k} = M_{1\ldots i-1,k}$ for such k, we find that

$$\{i,\ldots,e-1\} = \{j_i,\ldots,j_{e-1}\} \quad \text{and} \quad \{e,\ldots,n\} = \{j_e,\ldots,j_n\} \tag{5.73}$$

Let now $x,\ y \in \{i,\ldots,e-1\}$, then we find that

$$M_{{j_x}^{\pi},{j_y}^{\pi}} = M_{{j_x}^{\rho\pi},{j_y}^{\rho\pi}}, \tag{5.74}$$

according to (5.70).

For each $h \in U^{(i)}\,\rho\,\pi$ there exists a unique $g \in U^{(i)}\,\pi$ such that $h = g\,\pi^{-1}\,\rho\,\pi$. Because $g\,M > M$, there exists an $s \in \{i+1,\ldots,n\}$ such that $(g\,M)_{s-1} = M_{s-1}$ and $(g\,M)_s > M_s$. Hence as $g \in U^{(i-1)}\,\pi$ and $\mathsf{err}^{\pi}_{i-1}(M) = e$, we find that $s \leq e$. Indeed, if $s > e$ then $(g\,M)_e = M_e$ and then for all $j \in \{i,\ldots,e\}$ we have

$$\mathsf{tu}_{i-1}(M)_{j-(i-1)} = M_{1\ldots i-1,j} = (g\,M)_{1\ldots i-1,j} = \mathsf{tu}^{\pi}_{i-1}(M)_{j-(i-1)} \tag{5.75}$$

which contradicts $\mathsf{err}_{i-1}^{\pi}(M) = e$. Hence for each $j \in \{i, \ldots, s-1\}$ we find that

$$j^g,\ j^h \in V = \{{j_i}^{\pi}, \ldots, {j_{e-1}}^{\pi}\} \tag{5.76}$$

Hence for each $k \in \{i, \ldots, s-1\}$ we may write $k^g = {j_{x_k}}^{\pi} \in V$. And then

$$k^h = k^{g\,\pi^{-1}\,\rho\,\pi} = ({j_{x_k}}^{\pi})^{\pi^{-1}\,\rho\,\pi} = {j_{x_k}}^{\rho\,\pi} \tag{5.77}$$

Moreover the following relations hold:

- Let $k,\ m \in \{1, \ldots, i-1\}$ then as $g,\ h \in U^{(i-1)}\,\pi$ we have
$$(g\,M)_{k,m} = (h\,M)_{k,m}. \tag{5.78}$$
- Let $k \in \{i, \ldots, e-1\}$, $m \in \{1, \ldots, i-1\}$, and $k^g = {j_{x_k}}^{\pi} \in V$, then
$$\begin{aligned}(g\,M)_{k,m} &= M_{k^g,m^g} = M_{{j_{x_k}}^{\pi},m^{\pi}}\\ &= M_{{j_{x_k}}^{\rho\pi},m^h} = M_{k^h,m^h}\\ &= (h\,M)_{k,m}\end{aligned} \tag{5.79}$$
since $g,\ h \in U^{(i-1)}\,\pi$ and using (5.77).
- Let $k,\ m \in \{i, \ldots, e-1\}$, and then $k^g = {j_{x_k}}^{\pi} \in V$ and $m^g = {j_{x_m}}^{\pi} \in V$, then
$$\begin{aligned}(g\,M)_{k,m} &= M_{k^g,m^g} = M_{{j_{x_k}}^{\pi},{j_{x_m}}^{\pi}}\\ &= M_{{j_{x_k}}^{\rho\pi},{j_{x_m}}^{\rho\pi}} = M_{k^h,m^h}\\ &= (h\,M)_{k,m}\end{aligned} \tag{5.80}$$
using (5.77).

Hence for each $k, m \in \{1, \ldots, s-1\}$ we have $(g\,M)_{k,m} = (h\,M)_{k,m}$ and therefore

$$(h\,M)_{s-1} = (g\,M)_{s-1} = M_{s-1}. \tag{5.81}$$

Since $(g\,M)_s > M_s$ we have $(g\,M)_{1..s-1,s} > M_{1...s-1,s}$. We distinguish between the following cases:

- If $s^g \in V$, then by (5.78–5.80) we have
$$M_{1...s-1,s} < (g\,M)_{1...s-1,s} = (h\,M)_{1...s-1,s}. \tag{5.82}$$

- Otherwise, if $s^g \notin V$, say $s^g = {j_x}^\pi$ with $x \in \{e,\dots,n\}$, then $s^h = {j_y}^\pi$ with $y \in \{e,\dots,n\}$ and we find that
$$M_{1\dots p,s} \leq M_{1\dots p,e} < (\pi\, M)_{1\dots p,j_y} = (h\, M)_{1\dots p,s} \tag{5.83}$$

Hence by (5.81) and (5.82–5.83), we find that $(h\, M)_s > M_s$. ■

Theorem 5.5.7. *Let $M \in \mathcal{M}_n$ be lexically ordered and let $i \in \{2,\dots,n-2\}$ and $\pi \in \mathrm{Sym}(n)$ such that*
$$e = \mathsf{err}^\pi_{i-1}(M) > i, \quad \mathsf{act}^\pi_{i-1}(M) = 1, \quad \textit{and} \quad (\pi\, M)_{e-1} = M_{e-1}.$$
Let $\sigma \in U^{(i-1)}\,\pi$ such that $i^\sigma = k^\pi$ for some $k \in i^{(\mathrm{Aut}\, M_{e-1})^{(i-1)}}$. If $g\, M > M$ for each $g \in U^{(i)}\,\pi$, then $h\, M > M$ for each $h \in U^{(i)}\,\sigma$.

Proof : Consider $\tau \in (\mathrm{Aut}\, M_{e-1})^{(i-1)}$ such that $i^\tau = k$. Let ρ be the extension of τ to $\{1,\dots,n\}$ (with $j^\rho = j$ for each $j \in \{e,\dots,n\}$). Hence $\tau\, M_{e-1} = (\rho\, M)_{e-1} = M_{e-1}$. Therefore we find that $h\, M > M$ for each $h \in U^{(i)}\,\sigma$, according to Theorem 5.5.6. After all, $\rho\,\pi$ and σ have the same i-prefix. ■

Theorem 5.5.7 plays a central role in some of the improvements Algorithm 5.9 makes on Algorithm 5.8. Write $e = \mathsf{err}^\pi_{i-1}(M)$ and $p = \mathsf{piv}^\pi_{i-1}(M)$. Recall that the functions **rightCoset**(π, i ,s) and **rightCosetLast**(π, i ,s) check whether a permutation $h \in U^{(i-1)}\,\pi \subset U^{(s)} - U^{(s+1)}$ can be found such that $h\, M < M$. If $\mathsf{err}^\pi_{i-1}(M) \leq i+1$ and $\mathsf{act}^\pi_{i-1}(M) = 1$, then the action to be taken is to discard $U^{(i-1)}\,\pi$ entirely because $g\, M > M$ for each $g \in U^{(i-1)}\,\pi$. In this particular case Algorithm 5.9 now systematically invokes the method **subAutomo**. More precisely, we distinguish between the following two cases:

- If $\mathsf{err}^\pi_{i-1}(M) = i$, then we have $(\pi\, M)_{e-1} = M_{e-1}$. Moreover, for each $j \in \{\mathsf{piv}^\pi_{i-1}(M),\dots,i-2\}$ we find that $\mathsf{err}^\pi_j(M) = e$ and $\mathsf{act}^\pi_j(M) = 1$, according to Theorem 5.4.17 and Corollary 5.4.1. Hence, according to Theorem 5.5.7 we find for each such j that $h\, M > M$ where $h \in U^{(j+1)}\,\sigma$ such that $\sigma \in U^{(j)}\,\pi$, $(j+1)^\sigma = k^\pi$ and $k \in (j+1)^{(\mathrm{Aut}\, M_{e-1})^{(j)}}$. Clearly no such right coset $U^{(j+1)}\,\sigma$ should ever be considered in the remainder of the traversal of $U^{(p)}\,\pi$. So as to avoid the traversal of these right cosets, Algorithm 5.9 invokes the method **subAutomo**(π, i) (❼ in Algorithm 5.9). For each $j \in \{\mathsf{piv}^\pi_{i-1}(M),\dots,i-2\}$, this method adds the mark k^π to the set C_{j+1} when $k \in (j+1)^{(\mathrm{Aut}\, M_{e-1})^{(j)}}$.

- If $\mathsf{err}^{\pi}_{i-1}(M) = i+1$, then $(\pi' M)_{e-1} = M_{e-1}$ with $\pi' = (i\ l^{\pi^{-1}})$ and $\mathsf{lex}^{\pi}_{i-1}(M)_{[1]} = \{l\}$. Note that $\mathsf{err}^{\pi'}_{i}(M) = e$, $\mathsf{piv}^{\pi'}_{i}(M) = p$, $\mathsf{act}^{\pi'}_{i}(M) = 1$ and $g' M > M$ for each $g' \in U^{(i)}\,\pi'$. Hence similarly as in the first case, Algorithm 5.9 invokes the method **subAutomo**(π', i) (❻ in Algorithm 5.9).

Once we have discarded the right coset $U^{(i-1)}\,\pi$, Algorithm 5.9 uses the stored marks in such a way that when in the remainder of the traversal of $U^{(p)}\,\pi$ we are about to consider a right coset $U^{(j+1)}\,\tau$ with $j \in \{p, \ldots, i-2\}$ and $\tau \in U^{(j)}\,\pi$ such that $(j+1)^{\tau} \in C_{j+1}$, we can discard this right coset entirely (❹ in Algorithm 5.9).

While checking the canonicity of $M \in \mathcal{M}_n$, the incorporation of this technique into our canonicity algorithm, requires us to check whether $j \in (j+1)^{(\mathrm{Aut}\, M_{e-1})^{(j)}}$ where $e \in \{3, \ldots, n-1\}$ and $j \in \{0, \ldots, e-2\}$ (❽ in Algorithm 5.9). In order to do so these particular orbits need to be stored during earlier canonicity test on each of the leading principal submatrices of M. This can easily be done using the orbit partition we already maintain during each such canonicity test.

Finally, the question remains whether we still obtain, as in Algorithm 5.8, a strong generating set $S = S^{(0)}$ for the automorphism group $\mathrm{Aut}\, M$ with base $[1, \ldots, n]$ whenever M is minimal. As stated, the first technique only tries to guess some of the generators of S before we actually encounter them during traversal. Moreover, the second and third technique only discard right cosets which are known not to contain any automorphism of M. Hence when M is minimal, Algorithm 5.9 still obtains a strong generating set $S = S^{(0)}$ for the automorphism group $\mathrm{Aut}\, M$ with base $[1, \ldots, n]$.

5.5.3 Analysis and empirical data

The effectiveness of using the minimality and automorphism group of leading principal submatrices is examined by comparing data obtained from the orderly generation of the same classes of graphs as in Section 5.2.3. In Table 5.6, Algorithm 5.5, 5.8 and 5.9 are compared. The total number of graphs along with the total number of permutations which are checked during all executed canonicity tests is given. The empirical data illustrates that both techniques systematically reduce the number of permutations which are checked during orderly generation.

Algorithm 5.9 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function isCanonical($M \in \mathcal{M}_n$) : boolean
1: for $i \leftarrow n-1, \ldots 1$ do
2:   if guessGen($i$) $\vee$
       diffStab($i$) $< 0$ then                                   ❶
3:     return false
4: return true

function guessGen($i$ : int) : boolean
1: for all $\pi \in \bar{S}^{(i-1)}_{\mathrm{Aut}\, M_{n-1}}$ do
2:   if $i^\pi = \min((i^\pi)^{S^{(i-1)}_{\mathrm{Aut}\, M}})$ then
3:     if $M_{i^\pi \ldots (n-1)^\pi, n} = M_{i \ldots n-1, n}$ then
4:       $\bar{S}^{(i-1)}_{\mathrm{Aut}\, M} \leftarrow \bar{S}^{(i-1)}_{\mathrm{Aut}\, M} \cup \pi_{typeI}$          ❷
5:     else if $M_{i^\pi \ldots (n-1)^\pi, n} < M_{i \ldots n-1, n}$ then
6:       return true
7: for all $\pi \in \bar{S}^{(i-1)}_{\mathrm{Aut}\, M_{n-2}}$ do
8:   if $i^\pi = \min((i^\pi)^{S^{(i-1)}_{\mathrm{Aut}\, M}})$) then
9:     if $M_{1^\pi \ldots (n-2)^\pi, n} = M_{1 \ldots n-2, n-1}$ then
10:      if $M_{1^\pi \ldots (n-2)^\pi, n-1} = M_{1 \ldots n-2, n}$ then
11:        $\bar{S}^{(i-1)}_{\mathrm{Aut}\, M} \leftarrow \bar{S}^{(i-1)}_{\mathrm{Aut}\, M} \cup \pi_{typeII}$       ❸
12:      else if $M_{1^\pi \ldots (n-2)^\pi, n-1} < M_{1 \ldots n-2, n}$ then
13:        return true
14:    else if $M_{1^\pi \ldots (n-2)^\pi, n} < M_{1 \ldots n-2, n-1}$ then
15:      return true
16: return false

method subAutomo($\pi \in \mathrm{Sym}(n)$, $i$ : int)
1: $e \leftarrow \mathsf{err}^\pi_{i-1}(M)$
2: for $j \leftarrow \mathsf{piv}^\pi_{i-1}(M), \ldots, i-2$ do
3:   for all $k \in (j+1)^{(\mathrm{Aut}\, M_{e-1})^{(j)}}$ do                   ❽
4:     $C_{j+1} \leftarrow C_{j+1} \cup \{k^\pi\}$
```

Algorithm 5.9 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function diffStab($i$ : int) : int
 1: for all $j \in (\mathsf{lex}_{i-1}(M)_{[1]} \setminus \{i\})$ do
 2:   if $j = \min(j^{S^{(i-1)}_{\mathrm{Aut}M}})$ then
 3:     refine($(i\,j)$, $i$)
 4:     if $j \neq n$ then
 5:       $d \leftarrow$ rightCosetLast($(i\,j)$, $i+1$, $i-1$)
 6:     else
 7:       $d \leftarrow$ rightCoset($(i\,j)$, $i+1$, $i-1$)
 8:     if $d < 0$ then
 9:       return $d$
10: return 1

function rightCoset($\pi \in U^{(s)} - U^{(s+1)}$, $i, s$ : int) : int
 1: if checkErrorLevel($\pi$, $i$, $s$) then
 2:   return $\mathsf{act}^{\pi}_{i-1}(M)$
 3: else
 4:   $C_i \leftarrow \emptyset$
 5:   for all $j \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$ do
 6:     if $j \notin C_i$ then                                  ❹
 7:       $\pi' \leftarrow (i\ j^{\pi^{-1}})\ \pi$
 8:       refine($\pi'$, $i$)
 9:       $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
10:       if $d < 0 \vee d = 0$ then
11:         return $d$
12:       else
13:         subTranspose($\pi'$, $i$, $s$)                       ❺
14:   return 1
```

Algorithm 5.9 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function rightCosetLast($\pi \in U^{(s)} - U^{(s+1)}$, $i, s$ : int) : int
 1: if checkErrorLevel($\pi$, $i$, $s$) then
 2:    return $\mathsf{act}^{\pi}_{i-1}(M)$
 3: else if checkLast($i$) then
 4:    return 1
 5: else
 6:    $C_i \leftarrow \emptyset$
 7:    for all $j \in \mathsf{lex}^{\pi}_{i-1}(M)_{[1]}$ do
 8:       if $j \notin C_i$ then                                  ❹
 9:          $\pi' \leftarrow (i\ j^{\pi^{-1}})\ \pi$
10:          refine($\pi'$, $i$)
11:          if $j \neq n$ then
12:             $d \leftarrow$ rightCosetLast($\pi'$, $i+1$, $s$)
13:          else
14:             $d \leftarrow$ rightCoset($\pi'$, $i+1$, $s$)
15:          if $d < 0 \vee d = 0$ then
16:             return $d$
17:          else
18:             subTranspose($\pi'$, $i$, $s$ )                  ❺
19:    return 1

method subTranspose($\pi \in \mathrm{Sym}(n)$, $i, s$ : int)
 1: $e \leftarrow \mathsf{err}^{\pi}_{i}(M)$, $p \leftarrow \mathsf{piv}^{\pi}_{i}(M)$
 2: $j \leftarrow i-1$
 3: while $j > \max(s+1, p) \wedge (j\ j+1) \in \mathrm{Aut}\, M_{e-1}$ do
 4:    $C_j \leftarrow C_j \cup \{i^{\pi}\}$
 5:    $j \leftarrow j-1$
```

Algorithm 5.9 Checks whether $M \in \mathcal{M}_n$ is in column order canonical form.

```
function checkErrorLevel($\pi \in U^{(s)} - U^{(s+1)}, i, s$ : int) : boolean
1:  if $\mathsf{err}^{\pi}_{i-1}(M) \le i+1$ then
2:    if $\mathsf{err}^{\pi}_{i-1}(M) = n+1$ then
3:      $\bar{S}^{(s)}_{\mathrm{Aut}\,M} \leftarrow \bar{S}^{(s)}_{\mathrm{Aut}\,M} \cup \pi$
4:    else if $\mathsf{act}^{\pi}_{i-1}(M) = 1$ then
5:      if $\mathsf{err}^{\pi}_{i-1}(M) = i+1$ then
6:        $\pi' \leftarrow (i\ j^{\pi^{-1}})\pi$ with $\mathsf{lex}^{\pi}_{i-1}(M)_{[1]} = \{j\}$
7:        subAutomo($\pi'$, $i$)                                      ❻
8:      else
9:        subAutomo($\pi$, $i$)                                       ❼
10:   return true
11: else
12:   return false

function checkLast($\pi \in \mathrm{Sym}(n)$, $i$ : int) : boolean
1:  $e_{i-1} \leftarrow \mathsf{err}^{\pi}_{i-1}(M)$
2:  return $(\mathsf{act}^{\pi}_{i-1}(M) = 1 \wedge M_{1\ldots i-1,e_{i-1}} < M_{1^{\pi}\ldots(i-1)^{\pi},n}) \vee$
           $(\mathsf{act}^{\pi}_{i-1}(M) = -1 \wedge M_{1\ldots i-1,e_{i-1}} \le M_{1^{\pi}\ldots(i-1)^{\pi},n})$
```

					Algorithm 5.5	Algorithm 5.8	Algorithm 5.9
v	k	λ	μ	$\exists$	No. of perm.	No. of perm.	No. of perm.
5	2	0	1	1	15	15	7
9	4	1	2	1	123	98	57
10	3	0	1	1	161	133	95
	6	3	4		286	174	87
13	6	2	3	1	737	459	425
15	6	1	3	1	1 172	733	431
	8	4	4		1 731	659	293
16	5	0	2	1	2 762	1 048	371
	10	6	6		4 530	1 199	578
16	6	2	2	2	2 953	1 568	895
	9	4	6		5 600	3 132	2 244
17	8	3	4	1	3 751	1 815	1 664
21	10	3	6	1	6 966	3 594	1 568
	10	5	4		14 205	3 457	1 072
25	8	3	2	1	35 626	13 296	1 231
	16	9	12		211 670	87 601	63 330
25	12	5	6	15	2 027435	888 703	520 977
26	10	3	4	10	714 234	298 991	159 233
	15	8	9		18 139 933	5 961 926	3 600 190
27	10	1	5	1	95 807	27 456	5 931
	16	10	8		231 653	33 770	11 392
28	12	6	4	4	338 610	52 808	29 220
	15	6	10		256 250	112 382	86 010
29	14	6	7	41	203 348 967	74 443 194	68 443 130
36	10	4	2	1	1 194 185	384 989	3 388
	25	16	20		9 066 020	3 121 180	1 853 703
36	14	4	6	180	1 415 391 817	323 762 600	304 992 125
36	14	7	4	1	1 114 843	195 496	4 910
	21	10	15		524 654	225 287	48 432
40	12	2	4	28	8 787 084 036	567 121 307	414 216 280
45	16	8	4	1	15 685 156	2 064 434	9 077
	28	15	21		6 755 242	2 635 462	807 244
50	7	0	1	1	331 518 926	74 210 823	8 222 217
105	32	4	12	1	592 203 059	63 654 574	11 959 857

Tabel 5.6: Comparison of Algorithm 5.5, 5.8 and 5.9 applied in an orderly algorithm for strongly regular graphs

6 Case studies

”Forty-two!ïjelled Loonquawl. ”Is that all you’ve got to show for seven and a half million years’ work?I checked it very thoroughly,”said the computer, änd that quite definitely is the answer.”[D. ADAMS,THE HITCHHIKER’S GUIDE TO THE GALAXY]

As classification results are themselves of mathematical interest, we present in this final chapter several case studies together with new complete classifications results on association schemes, strongly regular and distance regular graphs obtained using (variants of) the generation algorithms as described in Chapters 4 and 5. Some of these new classification results appear in [21, 22, 30, 31], while other classification results still remain to be published. Note that the results in [22, 30] are listed in [14].

Attacking computational hard classification problems sometimes requires to consider and tackle such problems as individual instances. After all, often an algorithm that applies to a general class of combinatorial classification problems, will not be feasible for some particular instances. If the latter applies, we outline for each such instance the principal modifications incorporated into the general generation algorithm.

The remainder of this chapter is organized as follows. In Section 6.1 we present new (complete) classification results on strongly regular graphs. Furthermore, a number of earlier classification results on strongly regular graphs are verified. In [101], E. Van Dam lists all feasible parameter sets for three-class association schemes on at most 100 vertices. In Section 6.2 we report on new classification results for several cases in this list which were still open. Section 6.3 focuses on the classification of the Perkel graph, a primitive distance regular graph – which is also listed in [101] – for which the question of existence was not completely solved.

Remark 6.1. Note that the author of this dissertation is responsible for the design and implementation of all generation algorithms applied to obtain the classification results listed in this chapter. Mathematical interpretations of these classification results are largely due to the promotor of this dissertation. However, for matters of completeness, we sometimes opt to incorporate these interpretations into this chapter. •

6.1 Strongly regular graphs

In this section we present some new classification results on strongly regular graphs. Some of these classification results appear in [22, 30].

6.1.1 Introduction

Recall that a strongly regular graph with parameters (v, k, λ, μ), abbreviated by $\mathrm{srg}(v, k, \lambda, \mu)$, is a graph on v vertices that is regular of degree k such that each pair of adjacent vertices has λ common neighbours, and each pair of non-adjacent vertices has μ common neighbours. The intersection matrices L_1 and L_2 are defined by (2.17) which allow us to compute the eigenmatrix P and the dual eigenmatrix Q which are defined by (2.31) and (2.33), respectively. For every $i \in \{0, 1, 2\}$, the matrix entry P_{1i} is an eigenvalue of the adjacency matrix A with multiplicity Q_{0i}. From this it is easily seen that the adjacency matrix A has three distinct eigenvalues, namely, $\theta_0 = k$ of multiplicity $m_0 = 1$ and two others, $\theta_1 > \theta_2$, say, which are the solutions of $x^2 + (\mu - \lambda)x + (\mu - k) = 0$, of multiplicities m_1 and m_2, determined by the equations $1 + m_1 + m_2 = v$ and $k + m_1\theta_1 + m_2\theta_2 = 0$. Table 6.1 lists the eigenvalues and multiplicities of the adjacency matrix of the strongly regular graphs under consideration in this chapter (see also [14]). The adjacency matrix A of a strongly regular graph has some extra interesting algebraic properties. Recall that for each eigenvalue θ_i we may define a corresponding minimal idempotent matrix E_i which is positive semidefinite and has rank m_i. E_0 is the all-1 matrix J and E_1 and E_2 can be computed as follows:

$$E_1 = \frac{(A - \theta_0\, I)(A - \theta_2\, I)}{(\theta_1 - \theta_0)(\theta_1 - \theta_2)} \qquad E_2 = \frac{(A - \theta_0\, I)(A - \theta_1\, I)}{(\theta_2 - \theta_0)(\theta_2 - \theta_1)} \tag{6.1}$$

v	k	λ	μ	$\theta_1^{\mathbf{m_1}}$	$\theta_2^{\mathbf{m_2}}$	v	k	λ	μ	$\theta_1^{\mathbf{m_1}}$	$\theta_2^{\mathbf{m_2}}$
5	2	0	1	$\frac{-1+\sqrt{5}}{2}^{\mathbf{2}}$	$\frac{-1-\sqrt{5}}{2}^{\mathbf{2}}$	36	10	4	2	$4^{\mathbf{10}}$	$-2^{\mathbf{25}}$
9	4	1	2	$1^{\mathbf{4}}$	$-2^{\mathbf{4}}$	36	14	4	6	$2^{\mathbf{21}}$	$-4^{\mathbf{14}}$
10	3	0	1	$1^{\mathbf{5}}$	$-2^{\mathbf{4}}$	36	14	7	4	$5^{\mathbf{8}}$	$-2^{\mathbf{27}}$
13	6	2	3	$\frac{-1+\sqrt{13}}{2}^{\mathbf{6}}$	$\frac{-1-\sqrt{13}}{2}^{\mathbf{6}}$	36	15	6	6	$3^{\mathbf{15}}$	$-3^{\mathbf{20}}$
15	6	1	3	$1^{\mathbf{9}}$	$-3^{\mathbf{5}}$	40	12	2	4	$2^{\mathbf{24}}$	$-4^{\mathbf{15}}$
16	5	0	2	$1^{\mathbf{10}}$	$-3^{\mathbf{5}}$	45	12	3	3	$3^{\mathbf{20}}$	$-3^{\mathbf{24}}$
16	6	2	2	$2^{\mathbf{6}}$	$-2^{\mathbf{9}}$	45	16	8	4	$6^{\mathbf{9}}$	$-2^{\mathbf{35}}$
17	8	3	4	$\frac{-1+\sqrt{17}}{2}^{\mathbf{2}}$	$\frac{-1-\sqrt{17}}{8}^{\mathbf{8}}$	49	16	3	6	$2^{\mathbf{32}}$	$-5^{\mathbf{16}}$
21	10	3	6	$1^{\mathbf{14}}$	$-4^{\mathbf{6}}$	50	7	0	1	$2^{\mathbf{28}}$	$-3^{\mathbf{21}}$
25	8	3	2	$3^{\mathbf{8}}$	$-2^{\mathbf{16}}$	64	18	2	6	$2^{\mathbf{45}}$	$-6^{\mathbf{18}}$
25	12	5	6	$2^{\mathbf{12}}$	$-3^{\mathbf{12}}$	76	30	8	14	$2^{\mathbf{57}}$	$-8^{\mathbf{18}}$
26	10	3	4	$2^{\mathbf{13}}$	$-3^{\mathbf{12}}$	96	38	10	18	$2^{\mathbf{76}}$	$-10^{\mathbf{19}}$
27	10	1	5	$1^{\mathbf{20}}$	$-5^{\mathbf{6}}$	105	32	4	12	$2^{\mathbf{84}}$	$-10^{\mathbf{20}}$
28	12	6	4	$4^{\mathbf{7}}$	$-2^{\mathbf{20}}$	126	50	13	24	$2^{\mathbf{105}}$	$-13^{\mathbf{20}}$
29	14	6	7	$\frac{-1+\sqrt{29}}{2}^{\mathbf{14}}$	$\frac{-1-\sqrt{29}}{2}^{\mathbf{14}}$	176	70	18	34	$2^{\mathbf{154}}$	$-18^{\mathbf{21}}$
35	16	6	8	$2^{\mathbf{20}}$	$-4^{\mathbf{14}}$	253	112	36	60	$2^{\mathbf{230}}$	$-26^{\mathbf{22}}$

Tabel 6.1: Eigenvalues θ_1 and θ_2 with multiplicities m_1 and m_2 of the adjacency matrix of strongly regular graphs.

A more classic alternative for our positive semidefiniteness method is based on the *interlacing* of eigenvalues of A (a very good reference for which is [52, 55]). It is a consequence of this interlacing property that every principal submatrix P of A has all its eigenvalues except the largest one lying in the interval $[\theta_2, \theta_1]$. Moreover, if P has order n it must have an eigenvalue θ_2 with multiplicity at least $n - m_2 - 1$ and an eigenvalue θ_1 with multiplicity at least $n - m_1 - 1$.

As stated in Proposition 2.5.2, to avoid trivial examples we assume several conditions on the parameters v, k, λ, μ. The question of existence, and ultimately, classification, of strongly regular graphs satisfying these feasibility conditions has occupied the minds of many over the years. The smallest case for which these conditions are satisfied, but for which no strongly regular graph exists, is given by $(21, 10, 5, 4)$. At present all strongly regular graphs on $v \leq 36$ vertices are known. See, for example [13, 14] in conjunction with [72]. In addition to those whose uniqueness can be established without the use of a computer, there are some sporadic cases where the classification is complete. See [53], [95] and [96]. The smallest set of feasible parameters (v, k, λ, μ) for which the existence of the corresponding strongly regular graph is currently on doubt is $(65, 32, 15, 16)$. This is a particular example of the so-called half case.

6.1.2 Verification known results

As generation algorithms present answers to precise mathematical questions, these algorithms should be well-tested. Each design decision outlined in Chapter 4 and 5 was systematically tested against a list of known classification results for strongly regular graphs. The corresponding (v, k, λ, μ) parameter sets are listed in Table 6.2. The column labeled "CPU-time" gives the runtime of the general classification algorithm, while the column labeled "No. nodes recursion" indicates the size of the corresponding recursion tree. These results were obtained on a single CPU with a clock speed of approximately 3 GHz. In the comments column, references to earlier enumeration results are listed.

6.1.3 Strongly regular $(45, 12, 3, 3)$ graphs

The strongly regular $(45, 12, 3, 3)$ graphs are regular graphs on 45 points of degree 12 such that any two distinct points have 3 neighbours in common. One

v k λ μ	$\exists$ (cf. [14])	CPU-time	No. nodes recursion	Comment
5 2 0 1	1	0.09 s	10	Coolsaet, Fack [20]
9 4 1 2	1	0.12 s	36	Coolsaet, Fack [20]
10 3 0 1	1	0.14 s	45	Coolsaet, Fack [20]
13 6 2 3	1	0.17 s	87	Coolsaet, Fack [20]
15 6 1 3	1	0.16 s	131	Coolsaet, Fack [20]
16 5 0 2	1	0.16 s	120	Coolsaet, Fack [20]
16 6 2 2	2	0.21 s	211	Coolsaet, Fack [20]
17 8 3 4	1	0.22 s	285	Coolsaet, Fack [20]
21 10 3 6	1	0.22 s	215	Coolsaet, Fack [20]
25 8 3 2	1	0.24 s	408	Coolsaet, Fack [20]
25 12 5 6	15	4.20 s	61 799	Paulus [86]
26 10 3 4	10	1.80 s	39 882	Paulus [86]
27 10 1 5	1	0.20 s	441	Coolsaet, Fack [20]
28 12 6 4	4	0.55 s	3 074	
29 14 6 7	41	5 m 45 s	6 763 845	Bussemaker, Spence [97]
35 16 6 8	3854	7 h 18 m 8 s	360 810 530	McKay, Spence [72]
36 10 4 2	1	0.20 s	867	Coolsaet, Fack [20]
36 14 4 6	180	2 m 14 s	2 495 202	McKay, Spence [72]
36 14 7 4	1	0.30 s	1 666	
36 15 6 6	32648	80 h 40 m 8 s	7 432 568 114	McKay, Spence [72]
40 12 2 4	28	2 m 18 s	2 900 915	Spence [96, 98]
45 16 8 4	1	0.90 s	2 302	
49 16 3 6	0	53.20 s	884 431	Bussemaker, et al. [16]
50 7 0 1	1	3.50 s	1 950	
64 18 2 6	167	5 h 33 m 8 s	204 853 126	Haemers, Spence [53]

Tabel 6.2: Verification known classification result for srg(v, k, λ, μ)s with $v \leq 64$.

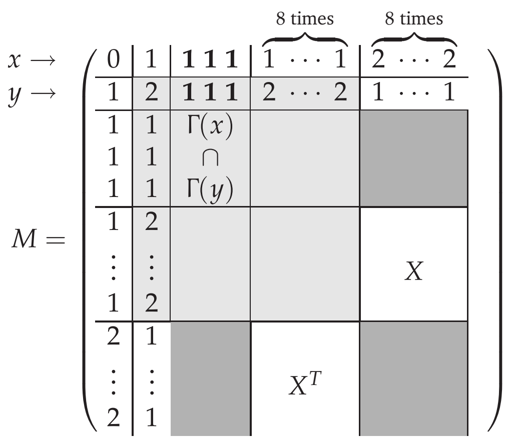
example of such an $\mathrm{srg}(45, 12, 3, 3)$ is provided by the point graph of the generalized quadrangle $\mathrm{GQ}(4, 2)$, and there are others already known, in fact 58 in total. These were found in [69] by R. Mathon and E. Spence in a search for such graphs with a particular block structure. In this section we report on two generation algorithms based on different techniques which have extended this list to a total of 78 pairwise non-isomorphic graphs, providing a complete classification. The second algorithm was designed and implemented independently by E. Spence. The complete classification has been published in [22].

The generation proceeds in several stages. In a preliminary stage both generation algorithms consider the plausible neighbourhood graphs $\Gamma(x)$ of a vertex x in an $\mathrm{srg}(45, 12, 3, 3)$. In what follows, we shall use the notation Δ for any such $\mathrm{srg}(45, 12, 3, 3)$. It is clear that a neighbourhood graph $\Gamma(x)$ has 12 vertices and is regular of degree 3. There exist (up to isomorphism) 94 regular graphs of order 12 and degree 3 (they can easily be generated using `geng` [77]). Of these, 21 can immediately be discarded as plausible neighbourhood graphs because they contain at least one pair of vertices that have more than two common neighbours (which together with x violate $\lambda = \mu = 3$).

In a further stage, the 73 matrices $N_1, \ldots N_{73}$ obtained in this way are extended to (symmetric) principal submatrices M of order 21 which correspond to possible subgraphs induced by $\Gamma(x) \cup \Gamma(y)$, where $y \in \Gamma(x)$. Without loss of generality we may reorder the rows and columns of M in such a way that M, with vertices $v_1, v_2, \ldots, v_{21}$, say, takes the form as depicted in Figure 6.1. Here $x = v_1$, $y = v_2$, $\Gamma(x)$ has vertices $v_2, v_3, \ldots, v_{13}$ and $\Gamma(y)$ has vertices $v_1, v_3, v_4, v_5, v_{14}, \ldots, v_{21}$. Thus, assuming $\Gamma(x)$ and $\Gamma(y)$ have been found, the only part of M that is unknown corresponds to the adjacencies between vertices $v_6, v_7, \ldots, v_{13}$ and $v_{14}, v_{15}, \ldots, v_{21}$, denoted by X in Figure 6.1.

We copy each of the matrices N_i in turn into the part of M which corresponds to the subgraph induced by $\Gamma(x)$ (cf. light grey part of Figure 6.1). In each case, we then generate all possibilities for the adjacencies between vertices $v_3, v_4, \ldots, v_{13}$ and vertices $v_{14}, v_{15}, \ldots, v_{21}$ and the mutual adjacencies between vertices $v_{14}, v_{15}, \ldots, v_{21}$ (cf. X and dark grey part of Figure 6.1).

The *first method* recursively fills the upper diagonal matrix entries of columns 14 to 21, column by column, starting with column 14. The combinatorial constraints and corresponding look-ahead strategy outlined in Section 4.1.1 together



Figuur 6.1: General form of the principal submatrix induced by $\Gamma(x) \cup \Gamma(y)$.

with the algebraic constraints outlined in Section 4.1.4 were used to constrain this search process. Furthermore, this first method also involves a pruning criterion based on the maximum clique size of Δ. From $\lambda = 3$ it follows that Δ can have cliques of size at most 5, (and certainly must have one of size 3) and hence that neighbourhood graphs in Δ can have cliques of size at most 4. We list the number of graphs N_i with a given clique size in Table 6.3.

Maximum clique size	4	3	2
Number of graphs	5	50	18

Tabel 6.3: Maximum clique size of plausible neighbourhood graphs.

Since M is to be embedded in the relation matrix of an srg$(45, 12, 3, 3)$, if we assume that that adjacency matrix has maximum clique size s ($3 \leq s \leq 5$), we may also assume the same to be true for M. Moreover, we may assume that both x and y belong to a clique of maximum size. Thus both $\Gamma(x)$ and $\Gamma(y)$ will have maximum clique size $s - 1$. In the extension process to the matrix of order 21 we prune the search tree whenever we meet a clique which has a size larger than s.

Searching for such small cliques is not very expensive in time, because we already need to keep track of the number of common neighbours of any two points in order to check the intersection constraints listed in Section 4.1.1. We use a *look-back* strategy to further improve the clique search. Suppose that the choice of 1 for the matrix entry $(M)_{p,q}$ completes a clique of size $d + 1$ (with d the size of a maximum clique), which means that we are able to prune the search at this point. Then this choice remains forbidden, so that $M_{p,q} = 2$, until a backtrack occurs to the second last choice of matrix entry that contributed to the same clique.

We also make use of another isomorphism rejection technique: we fix an ordering of the list $N_1, \ldots, N_{73}$ in such a way that the maximum clique size decreases while we traverse the list from front to back. Then, whenever we have fully generated the submatrix $\Gamma(y)$ we determine using McKay's program `nauty` [71], the index of the unique graph in this list which is isomorphic to $\Gamma(y)$. If this index is smaller than the index of $\Gamma(x)$, we discard the result. Indeed, the matrix we obtain by interchanging x and y in this result will also be generated during the search and yields an isomorphic subgraph.

A final pruning criterion that avoids the construction of too many isomorphic submatrices M, is that we require for each $p, q \in \Gamma(y) \setminus \{\Gamma(x) \cup \{x\}\}$ with $p \leq q$ that the tuple $(M_{p,1}, \ldots, M_{p,21})$ is *lexicographically* smaller than or equal to the tuple $(M_{q,1}, \ldots, M_{q,21})$. This first method produced approximately $35, 000, 000$ pairwise non-isomorphic candidates M.

The *second method* of generating possible 21×21 submatrices differed essentially in only one way. When it came to filling in the entries of X in the matrix M of Figure 6.1, these were generated column by column, starting with column 14. Similar combinatorial constraints where used as in the first method, whereas the algebraic constraints were based on *interlacing* of eigenvalues of the adjacency matrix. However no look-ahead strategies or dynamical variable ordering techniques were applied. So as to check the interlacing of eigenvalues, an iterative procedure was used to determine eigenvalues to within a chosen error and with an upper bound to the number of iterations allowed. A candidate matrix whose eigenvalues were not determined within the maximum number of iterations was accepted even though its eigenvalues may well have been outside the required range. In the end this second method produced approximately $21, 000, 000$ pairwise non-isomorphic candidates M.

In the last stage, having produced a list of non-isomorphic candidate submatrices M, we attempt recursively to extend each of them to a full relation matrix R. Again we apply all the constraints listed in Sections 4.1.1 and 4.1.4. Most of the time the extension fails rather quickly. The application in this last stage of the look-ahead strategy and dynamic variable ordering techniques certainly contributes to this behaviour. Since most of the time the extension fails rather quickly and checking which of the N_i is isomorphic to $\Gamma(z)$ for a given $z \in \Gamma(x)$ is a rather costly operation, it turns out to be no longer necessary in this last stage to compare neighbourhood graphs of other points of $\Gamma(x)$ with the list $N_1, \dots, N_{73}$. We still prune on maximum clique sizes, because that check is fairly quick. As in the first stage we we require for each $p, q \notin \Gamma(y) \setminus \{\Gamma(x) \cup \{x\}\}$ with $p \leq q$ that the row $(R_{p,1}, \dots, R_{p,45})$ is lexicographically smaller or equal than the row $(R_{q,1}, \dots, R_{q,45})$. Finally we use `nauty` [71] to filter out isomorphic matrices R among the results obtained so that we retain exactly one representative from each isomorphism class.

Maximum clique size 5		Maximum clique size 4	
|Aut (srg)|	number of srgs	|Aut (srg)|	number of srgs
51840	1	162	1
1152	1	54	1
216	1	18	5
72	2	9	1
48	3	6	16
36	1	3	7
18	1	2	15
12	3	1	8
10	1		
8	3		
4	5		
2	2		
Total	24	Total	54

Tabel 6.4: Automorphism group size for each srg$(45, 12, 3, 3)$.

As stated, both computer programs independently proved that there are up to isomorphism exactly 78 different strongly regular graphs with parameters $(v, k, \lambda, \mu) = (45, 12, 3, 3)$, providing a complete classification. A file containing all of these graphs can be obtained from [97]. Of these 78 graphs, 24 graphs contain a clique of size 5. There are 54 graphs which do not contain a clique of size 5 but do have a clique of size 4. No srg$(45, 12, 3, 3)$ exists which does not contain a clique of size 4. In Table 6.4 we list these graphs and the sizes of their automorphism groups. The column labeled "number of srgs" gives the number of isomorphism classes of srg$(45, 12, 3, 3)$s whose automorphism group has the size indicated in the column labeled "$|\mathrm{Aut}\,(\mathrm{srg})|$".

It may be of interest to the reader to know the extent of the computational power and the computer languages that were used in the investigation. As far as for the first generation algorithm, the language used was Java and the computation done on a batch of 24 Linux-based AMD Athlon MP 1800 and 2600 PCs, while for the second generation algorithm, the computation was done on a twin processor, 2.2 GHz Linux-based workstation with gcc's C compiler. The first generation algorithm took less than 2 weeks CPU-time to complete. The second generation algorithm took substantially longer to complete. However, an exact record of the time taken was not kept.

Remark 6.2. The classification of the strongly regular $(45, 12, 3, 3)$ graphs was done at at time when the canonicity algorithm in Chapter 5 was still under development. Afterwards, this classification problem was reconsidered using the standard column orderly approach as described in Chapter 4 and 5, yielding the same 78 non-isomorphic srg$(45, 12, 3, 3)$s. •

6.1.4 Strongly regular subgraphs of the McLaughlin graph

The McLaughlin graph [46] is the well known unique strongly regular graph with $(v, k, \lambda, \mu) = (275, 112, 30, 56)$. This graph contains many induced subgraphs which are again strongly regular. For four of the corresponding (v, k, λ, μ) parameters sets, that is, $(105, 32, 4, 12)$, $(120, 42, 8, 18)$, $(176, 70, 18, 34)$ and $(253, 112, 36, 60)$, the strongly regular graph is not known to be unique. In this section we report on an exhaustive computer search which settles the uniqueness question in the first three cases. These classification results are accepted for publication in [30].

The general generation algorithm as outlined in Chapter 4 and 5 was used to tackle the above parameter sets. However, one additional criterion, based on

Lemma 6.1.1, is used to speed up the search process.

Lemma 6.1.1. *A graph Γ contains a clique of size s if and only if the column order canonical form of the corresponding matrix M has a leading principal submatrix of order $s \times s$ with all non-diagonal entries equal to* 1.

Proof : If the top left principal $s \times s$ submatrix of M has the stated form, then by definition of M the vertices of Γ numbered $1, \ldots, s$ are mutually adjacent and hence form a clique. If the leading principal $s \times s$ submatrix of M does not have the stated form, then $\mathsf{cert}_\mathsf{c}(M)$ contains at least one 2 within the first $s(s-1)/2$ positions. If then Γ has a clique of size s and we renumber the vertices of Γ in such a way that the clique vertices are numbered $1, \ldots, s$, we obtain a matrix M' for which $\mathsf{cert}_\mathsf{c}(M')$ starts with $s(s-1)/2$ ones. Hence $\mathsf{cert}_\mathsf{c}(M') < \mathsf{cert}_\mathsf{c}(M)$, and M is not in column order canonical form. ∎

We apply this lemma in the following way : if at a certain point in the search process the partially instantiated matrix M has a completely instantiated leading principal submatrix of order $s \times s$ which corresponds to a clique, while the leading principal submatrix of order $s+1 \times s+1$ does not, then we may prune the search if we detect a clique of size $s+1$ elsewhere in the graph. In general, searching for cliques in a graph is a costly operation. However for the three parameter sets we consider, the maximal clique size can be proved to be at most 4 (e.g., using the *Hoffman bound* [12]). Finally, to improve the clique search, we use a similar *look-back* strategy as in Section 6.1.3.

v	k	λ	μ	CPU time
105	32	4	12	5 s
120	42	8	18	13 m 40 s
176	70	18	34	11 h 45 m 10 s

Tabel 6.5: Runtime generation algorithm.

The main result of our computer search is that the strongly regular graphs with parameters $(105, 32, 4, 12)$, $(120, 42, 8, 18)$ and $(176, 70, 18, 34)$ are uniquely[1] determined by their (v, k, λ, μ) parameters (up to isomorphism). The generation

[1]Uniqueness of the strongly regular $(105, 32, 4, 12)$ graph was afterwards established theoretical in [23].

algorithm was implemented in the programming language Java and the search was carried out on a single CPU with a clock speed of approximately 1 GHz. Table 6.5 lists the time needed to perform the exhaustive searches.

6.1.5 Strongly regular $(126, 50, 13, 24)$ graphs

According to [56, Proposition 6.1], the Hermitian two-graph $\mathcal{H}(5)$ contains an $\mathrm{srg}(126, 50, 13, 24)$. Using the same generation algorithm as outlined in Section 6.1.4, we found that this strongly regular graph is uniquely determined by its (v, k, λ, μ) parameter set (up to isomorphism). The generation algorithm was implemented in the programming language Java and the search was carried out on a single CPU with a clock speed of approximately 1 GHz. It took less than 5 days CPU time to complete the search.

6.1.6 Strongly regular $(96, 38, 10, 18)$ graphs

It is an open problem whether a regular two-graph on 76 and 96 vertices exists [48, 99]. The strongly regular $(73, 30, 8, 14)$ and $(96, 30, 8, 18)$ graphs – if such graphs exist – belong to the switching class of a regular two-graph on 76 and 96 vertices exists, respectively. Using the same generation algorithm as outlined in Section 6.1.4, we found that no strongly regular graph $(96, 30, 8, 18)$ exists. The generation algorithm was implemented in the programming language Java and the search was done on a batch of 24 Linux-based AMD Athlon MP 1800 and 2600 PCs. It took 286 days CPU time to complete the search.

For the classification of the strongly regular $(76, 30, 8, 14)$ graphs, already more than 5 years CPU-time were used on a mixed batch of 24 Linux-based AMD Athlon MP 1800 and 2600 PCs and 12 Pentium D 3000 PCs., discarding more than 90 percent of 72548 plausible neighbourhood graphs. Unfortunately, no strongly regular graphs were found yet.

6.2 Three-class association schemes

In [101], E. Van Dam describes several constructions of three-class association schemes and gives theoretical conditions for existence of such three-class association schemes with given parameter sets. He also lists all feasible parameter sets for three-class association schemes on at most 100 vertices. In this section we report on a computer classification of several cases in this list which were still left open. Note that some of these classification results appear in [31].

The results listed below use a similar notation as in [101]. The first column gives the number of vertices, denoted by n. The second column lists the degrees k_i of the graphs for the corresponding relations R_i and columns L_1, L_2, L_3 denote the intersection matrices with top row and leftmost column omitted. The column marked # lists the number of known non-isomorphic association schemes of that type, as given in [101], and in **bold face** the number of non-isomorphic schemes as generated by our programs. The last column indicates some additional information specific to the three-class association schemes under consideration.

Several parameter sets of the three-class association schemes listed in [101] have in common that they satisfy the additional property that (at least) one of their relations, say R_s, has a corresponding graph G_s which must be strongly regular. In other words, if we identify the two remaining (nontrivial) relations, we always obtain a two-class association scheme. The fact that G_s is strongly regular and the parameters of G_s can be inferred from the intersection numbers of Ω. This extra property enables us to divide the generation process into several stages, as follows:

1. We start from a complete list of non-isomorphic strongly regular graphs Γ_s with the parameters of G_s.
2. For each element in this list we reduce the domain to the singleton $\{s\}$ for those pairs x, y that are adjacent in Γ_s.
3. We then execute an exhaustive generation process which only applies the combinatorial constraints and corresponding look-ahead strategy with dynamic variable ordering as outlined in Section 4.1.1, 4.1.2 and 4.1.3.
4. Finally, we remove isomorphic solutions from this result by an explicit isomorphism test based on the well-known program `nauty` [71].

The number of isomorphic three-class association schemes we obtain from a single strongly regular graph Γ_s can be at most the size $|\mathrm{Aut}\Gamma_s|$ of the automorphism group of this graph. In most cases this number is rather small, and always significantly smaller than the number of isomorphic elements we might expect if we started the generation from scratch. For this reason it was not necessary to incorporate an isomorphism removal technique in the algorithm proper.

Our approach was made possible because other authors have classified the corresponding strongly regular graphs Γ_s. Note however that each of these strongly regular graphs can also be generated by the generation algorithms developed in this text (cf. Section 6.1.2). The following table lists those results which were needed for our search. For each parameter set (v, k, λ, μ) of a strongly regular graph we also list the parameter set for the complement of that graph.

v	k	λ	μ	Nr. of graphs	Author(s)	References
35	16	6	8	3854	B.D. McKay, E. Spence	[72, 78]
35	18	9	9			[96, 97]
40	12	2	4	28	E. Spence	[78, 95, 97]
40	27	18	18			
45	12	3	3	78	K. Coolsaet, J. Degraer	[22, 97]
45	32	22	24		E. Spence	
64	18	2	6	167	W. Haemers, E.Spence	[53, 97]
64	45	32	30			

To increase the reliability of the computer results listed below, we have used a second (more classical) generation method for some of the parameter sets. This method requires a longer execution time than the first method. We have applied this second method in the special case where the three-class association scheme Ω corresponds to a *Hoffman-coloring* of G_s (again this is a property of the intersection numbers of Ω, not an additional requirement we impose). A Hoffman-coloring of G_s is a partition of the vertices of G_s into $1 + k/m$ disjoint cocliques of equal size, where $-m$ is the smallest eigenvalue of the graph (see [54]). The Hoffman-coloring defines an equivalence relation which turns out to be one of the relations R_h of the association scheme. Note that the structure of the graph G_h that corresponds to R_h is uniquely determined by the intersection numbers of Ω.

The second generation method runs along similar lines as the method discussed before but now it starts by adjusting the domains of the elements according to G_h instead of Γ_s. In this case we do not need to know the list of possible strongly graphs Γ_s in advance. Unfortunately the automorphism group $\mathrm{Aut}G_h$ is too large, so we need some extra orderly isomorphism removal techniques. Internally, the relation of the scheme are reordered in such a way that R_h corresponds to the 1-st associate class, because this allows us to use a column order canonical form during generation. We also introduce additional constraints based on the algebraic properties of the minimal idempotents of the association scheme as described in Section 4.1.4

In the results listed below, the last column indicates which relation corresponds to a strongly regular graph, and which relation is a Hoffman-coloring of that graph (when such a coloring exists). Two of the parameter sets have the additional property that one of the relations R_i defines a *distance regular graph* G_i of diameter 3 (an imprimitive distance regular cover of a complete graph). Again this is indicated in the last column (using the abbreviation 'drg').

A complete list of all resulting association schemes can be retrieved from the web page `http://caagt.UGent.be/tca/`.

n	k_i	L_1			L_2			L_3			#	
35	12	4	6	1	6	9	3	1	3	0	≥ 1	
	18	4	6	2	6	9	2	2	2	0	$\Downarrow$	$\mathrm{srg}(35,18,9,9)$
	4	3	9	0	9	9	0	0	0	3	**35**	coloring $7K_5$

The exhaustive search applied to the parameters listed above, yields 35 non-isomorphic three-class association schemes with this parameter set, where before only 1 example was known. These 35 association schemes arise from 22 different strongly regular graphs. The following table lists the sizes of the underlying automorphism groups: $\mathrm{Aut}\Gamma_s$ for the strongly regular graph and $\mathrm{Aut}\,\Omega$ for the related three-class association schemes.

$\lvert\mathrm{Aut}\Gamma_s\rvert$	$\lvert\mathrm{Aut}\Omega\rvert$
40320	168
288	24, 24
96	4
64	8,4
32	8,8,4,4
21	21
16	4
12	12,12
12	3
8	4,4
8	4,4
8	2
8	2
4	4,4
4	2,2
4	2,2
4	1
4	1
3	3,3
3	3
3	3
2	1,1

The six association schemes with largest automorphism groups ($\lvert\mathrm{Aut}\Omega\rvert \geq 12$) correspond to solutions of *Kirkman's schoolgirl problem*: how to find 7 arrangements (one for each day of the week) of a class of 15 schoolgirls in 5 groups of 3 such that at the end of the week every girl has shared a group with every other girl exactly once? The $7 \times 5 = 35$ groups are the vertices of the associationscheme, two groups are in relation R_2 if they have a girl in common, in relation R_3 if they belong to the same day and in relation R_1 otherwise. There are exactly seven non-isomorphic solutions to Kirkman's schoolgirl problem (see Table 1.1), leading to 6 different association schemes. (The two solutions where girls and groups correspond to points and lines of $\mathrm{PG}(3,2)$ yield the same association scheme.)

n	k_i	L_1			L_2			L_3			#	
40	9	2	6	0	6	18	3	0	3	0	≥ 1	drg
	27	2	6	1	6	18	2	1	2	0	$\Downarrow$	srg(40, 27, 18, 18)
	3	0	9	0	9	18	0	0	0	2	**3**	coloring 10 K_4

(G_1 is a distance regular antipodal cover of the complete graph K_{10}.)

There are 3 non-isomorphic three-class association schemes which satisfy this parameter set, where before only 1 example was known. They arise from two different strongly regular graphs. The table below lists the sizes of the corresponding automorphism groups: $\mathrm{Aut}\Gamma_s$ for the strongly regular graph and $\mathrm{Aut}\Omega$ for the related three-class association schemes.. The largest automorphism group corresponds to the association scheme generated from the symplectic generalized quadrangle $W(3)$ colored by the lines of a spread. For a construction of the other two, we refer to the paper [31].

$\lvert\mathrm{Aut}\Gamma_s\rvert$	$\lvert\mathrm{Aut}\Omega\rvert$
51840	1440
432	432,144

For a second set of intersection numbers for $n = 40$ it turns out that *no* three-class association scheme exists :

n	k_i	L_1			L_2			L_3			#	
40	9	0	4	4	4	4	4	4	4	10	≥ 0	
	12	3	3	3	3	2	6	3	6	9	$\Downarrow$	srg$(40, 12, 2, 4)$
	18	2	2	5	2	4	6	5	6	6	**0**	

n	k_i	L_1			L_2			L_3			#	
45	24	12	5	6	5	0	3	6	3	3	≥ 2	
	8	15	0	9	0	7	0	9	0	3	$\Downarrow$	coloring $5\,K_9$
	12	12	6	6	6	0	2	6	2	3	**764**	srg$(45, 12, 3, 3)$

There are 764 non-isomorphic three-class association schemes of the above type, where only 2 examples were known before. Automorphism groups range in size from 1 up to 1296. The table below lists the sizes of the corresponding automorphism groups. They arise from 75 different strongly regular graphs. There are 354 cases with a trivial automorphism group and the largest automorphism group corresponds to an association scheme that can be constructed from the unique generalized quadrangle with parameters $(s, t) = (4, 2)$.

$\lvert\mathrm{Aut}\Omega\rvert$	#	$\lvert\mathrm{Aut}\Omega\rvert$	#	$\lvert\mathrm{Aut}\Omega\rvert$	#
1296	1	36	3	6	105
162	1	18	36	4	6
144	1	16	2	3	106
108	3	12	12	2	107
54	9	10	1	1	354
48	2	9	15		

There are *no* three-class association schemes with the intersection numbers for $n = 45$ below. Note that for the first parameter set, the graph G_2 is also strongly regular (and isomorphic to the triangular graph $T(10)$). However, using that graph as a starting point for our algorithm takes too long to complete.

n	k_i	L_1			L_2			L_3			#	
45	16	6	6	3	6	4	6	3	6	3	≥ 0	
	16	6	4	6	4	8	3	6	3	3	$\Downarrow$	
	12	4	8	4	8	4	4	4	4	3	**0**	srg$(45, 12, 3, 3)$

n	k_i	L_1			L_2			L_3			#	
45	16	7	5	3	5	5	6	3	6	3	≥ 0	
	16	5	5	6	5	7	3	6	3	3	$\Downarrow$	
	12	4	8	4	8	4	4	4	4	3	**0**	srg$(45, 12, 3, 3)$

n	k_i	L_1			L_2			L_3			#	
64	15	2	12	0	12	30	3	0	3	0	≥ 5	drg
	45	4	10	1	10	32	2	1	2	0	$\Downarrow$	$(64, 45, 32, 30)$
	3	0	15	0	15	30	0	0	0	2	**94**	coloring 16 K_4

(G_1 is a distance regular antipodal cover of the complete graph K_{16}.)

There are 94 non-isomorphic three-class association schemes of the above type, where only 5 examples were known before. Automorphism groups range in size from 16 up to 23040. These 94 association schemes arise from 17 different strongly regular graphs. The table below lists the sizes of the underlying automorphism groups: $\mathrm{Aut}\Gamma_s$ for the strongly regular graph and $\mathrm{Aut}\Omega$ for the related three-class association schemes. Among the results we find the 5 association schemes that correspond to spreads in the unique generalized quadrangle with parameters $(s, t) = (3, 5)$. We have verified by computer that at least 80 of the generated schemes can be constructed by means of the method described by D.G Fon-Der-Flaass in [43, construction 4]. For a more detailed mathematical discussion of these 94 non-isomorphic three-class association schemes we refer the reader to [62, chapter 4].

$\lvert\mathrm{Aut}\Gamma_s\rvert$	$\lvert\mathrm{Aut}\Omega\rvert$
138240	23040, 1536, 1152, 384, 120
3072	768, 256, 64, 192
3072	1536, 512, 384
1536	768, 256, 192
384	384, 384, 384, 192, 96, 96, 96, 96, 96, 64
384	384, 384, 192, 96, 96, 64
288	288, 288, 288, 288, 96, 96, 96, 96, 72
256	128, 128, 64, 64, 64
256	128, 128, 64, 64, 64
192	96, 96, 96, 96, 96, 96, 48
128	64, 64, 32, 16
64	32, 32, 32, 32, 32, 32, 16
64	32
64	64, 64, 64, 64, 32, 32, 32, 32, 32, 16, 16
32	32, 32, 32, 32, 32, 32, 32, 32, 16, 16, 16, 16
32	32
32	32

n	k_i	L_1			L_2			L_3			#	
96	19	6	12	0	12	30	15	0	15	4	≥ 0	primitive drg
	57	4	10	5	10	36	10	5	10	4	$\Downarrow$	$(96, 57, 36, 30)$
	19	0	15	4	15	30	12	4	12	2	**0**	

n	k_i	L_1			L_2			L_3			#	
96	38	14	9	14	9	4	6	14	6	18	≥ 0	
	19	18	8	12	8	2	8	12	8	18	$\Downarrow$	
	38	14	6	18	6	4	9	18	9	10	**0**	$(96, 38, 10, 18)$

n	k_i	L_1			L_2			L_3			#	
96	19	4	12	2	12	30	15	2	15	2	≥ 0	
	57	4	10	5	10	36	10	5	10	4	$\Downarrow$	$(96, 57, 36, 30)$
	19	2	15	2	15	30	12	2	12	4	**0**	

Three feasible parameter sets for three-class association schemes have in common that they satisfy the additional property that (at least) one of their relations, say R_s, has a corresponding graph G_s which must be strongly regular with (v, k, λ, μ) parameters either $(96, 38, 10, 18)$ or $(96, 57, 36, 30)$. Since the non-existence of such strongly regular graphs was established in Section 6.1.6, non-existence follows immediately for each of these parameter sets. Note that the first feasible parameter set listed above, corresponds to a primitive distance regular graph of diameter 3 with intersection array $\{19, 2, 5; 1, 4, 15\}$. Non-existence of this primitive distance regular graph was afterwards established theoretical in [24].

n	k_i	L_1			L_2			L_3			#	
57	6	0	5	0	5	15	10	0	10	10	≥ 1	primitive drg
	30	1	3	2	3	4	12	2	12	6	$\Downarrow$	Perkel graph
	20	0	3	3	3	8	9	3	9	7	**1**	

n	k_i	L_1			L_2			L_3			#	
88	12	1	10	0	10	40	10	0	10	5	≥ 0	primitive drg
	60	2	8	2	8	40	11	2	11	2	$\Downarrow$	
	15	0	8	4	8	44	8	4	8	2	**0**	

The above two feasible parameter sets for three-class association schemes have in common that they correspond to primitive distance regular graphs of diameter 3, with intersection array $\{6, 5, 2; 1, 1, 3\}$ and $\{12, 10, 2; 1, 2, 8\}$, respectively. The classification of the parameter set with $n = 57$ is described in Section 6.3. The distance regular graphs on 88 vertices were generated using two different programs. In a first program, we used the general generation algorithm as outlined in Chapter 4 and 5. A second program uses the same approach as the first generation algorithm described in Section 6.3. Unfortunately, no such distance regular graphs exist.

The three feasible parameter sets for three-class association schemes listed below were classified using the general generation algorithm as outlined in Chapter 4 and 5.

n	k_i	L_1			L_2			L_3			#	
31	10	3	4	2	4	2	4	2	4	4	≥ 1	
	10	4	2	4	2	3	4	4	4	2	$\Downarrow$	
	10	2	4	4	4	4	2	4	2	3	**1**	

n	k_i	L_1			L_2			L_3			#	
42	13	4	8	0	8	16	2	0	2	0	≥ 1	drg
	26	4	8	1	8	16	1	1	1	0	$\Downarrow$	
	2	0	13	0	13	13	0	0	0	1	**2**	

The two association schemes found, have an automorphism group of size 2184 and 168. Only the association scheme with the largest automorphism group was previously known. Note that G_1 is a distance regular antipodal cover of the complete graph K_{14}.

n	k_i	L_1			L_2			L_3			#	
51	16	5	10	0	10	20	2	0	2	0	≥ 1	drg
	32	5	10	1	10	20	1	1	1	0	$\Downarrow$	
	2	0	16	0	16	16	0	0	0	1	**4**	

The four association schemes found, have an automorphism group of size 16320, 240, 192 and 48. Only the association scheme with the largest automorphism group was previously known. Note that G_1 is a distance regular antipodal cover of the complete graph K_{17}.

6.3 The Perkel graph

The *Perkel graph* is a distance-regular graph of order 57, degree 6 and diameter 3, with intersection array $\{6, 5, 2; 1, 1, 3\}$. We refer to [12] for several constructions of this graph and for the proof that it is distance-regular. This section describes a computer assisted proof that all distance-regular graphs with this intersection array are isomorphic. Note that the given intersection array is among the smallest for which this classification question is not yet settled [12, 15]. This classification result has been published in [21].

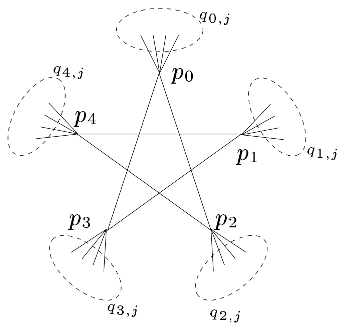
Define a *pseudo Perkel graph* to be a distance-regular graph with the same intersection array as the Perkel graph. In what follows we shall use the notation Γ for any such graph. The exhaustive generation process proceeds in two phases. In a first step we generate a set of plausible subgraphs of Γ with certain prescribed properties. In a second step we try to extend each of these results to a full pseudo Perkel graph. To increase the reliability of the computer proof we perform the generation process twice, choosing different subgraphs for the first phase. Note that the two methods were programmed by different authors as an added guard against programming errors. The programs were written in Java and run in parallel on 20 computers with CPUs of the Pentium III and Pentium IV type, and processor speeds ranging from 500 to 1600MHz.

For the first version we generate all plausible subgraphs Δ induced on the vertices of Γ at distance 3 of a fixed vertex ω. From the regularity properties of Γ we may derive certain regularity properties on Δ, that is, Δ is a regular graph of order $k_3 = 20$ and degree $p^3_{13} = 3$. Also, being a subgraph of Γ, Δ has girth at least 5. There are, up to isomorphism, 5783 connected graphs of order 20 and degree three and girth at least five [9] . We use a precomputed list of these graphs (downloaded from [88]). Note that Δ needs not be connected, hence in addition we also need to consider the disjoint union of two Petersen graphs as a candidate for Δ. Likewise the rank and positive semidefiniteness constraints on Γ give rise to rank and positive semidefiniteness constraints on Δ. More information can be found in [21]. The first step of the algorithm generated 105 different matrices N corresponding to 20 non-isomorphic graphs Δ. This step took approximately 4 million seconds of computer time (46 days). From these 20 graphs only a single one could be extended to a full graph, yielding the 'true' Perkel graph. This last step took about 7 seconds.

The second version uses the properties of the neighborhood of a pentagon in Γ. In this version the last step of the algorithm can be replaced by a purely mathematical proof [21]. Because $p^1_{22} > 0$, every pseudo Perkel graph must contain at least one pentagon. Consider such a pentagon, or more accurately, a pentagram $P = \{p_0, \ldots, p_4\}$ in Γ with adjacencies $p_i \sim p_{i\pm 2}$. (All index arithmetic is done modulo 5). Each vertex of P is adjacent to 4 vertices of $\Gamma - P$. We denote these vertices by q_{ij}, $i = 0, \ldots, 5, j = 0, \ldots, 3$ with $q_{ij} \sim p_i$ (cf. Figure 6.2). Because the girth of Γ is 5, no two of these vertices coincide and therefore the neighborhood $\Gamma(P)$ of P contains 20 vertices. We shall be interested in the subgraph Φ of Γ induced on the union $P \cup \Gamma(P)$.

The second algorithm generates all plausible graphs Φ and then tries to extend them to pseudo Perkel graphs. This exhaustive generation process is based on the following lemma :

Lemma 6.3.1. *Let p_i, q_{ij} be defined as above. Then q_{ij} can be adjacent to at most one vertex of the form $q_{i+1,k}$, and at most one vertex of the form $q_{i-1,n}$.*



Figuur 6.2: A pentagon of Γ and its neighborhood.

Vertices of the form q_{ij} and $q_{i\pm 2,k}$ are never adjacent. Moreover, for each i there are exactly two $j \in \{0,1,2,3\}$ such that $q_{ij} \sim q_{i+1,k}$ for some $k \in \{0,1,2,3\}$, and similarly, there are exactly two $m \in \{0,1,2,3\}$ such that $q_{im} \sim q_{i-1,n}$ for some $n \in \{0,1,2,3\}$.

Proof : If q_{ij} were adjacent to two vertices of the form $q_{i+1,k}$, then together with p_i these vertices would form a quadrangle, contradicting the girth of Γ (and similarly for $q_{i-1,m}$). If q_{ij} were adjacent to $q_{i+2,k}$ then we would obtain a quadrangle by adding the edge $p_i - p_{i+2}$ (and similarly for $q_{i-2,m}$). As $d_\Gamma(p_i, p_{i+1}) = 2$ $p^2_{12} = 3$, there must be exactly 3 paths of length 3 in Γ joining these vertices. There is one path $p_i \sim p_{i+2} \sim p_{i+4} \sim p_{i+1}$ that lies entirely in P, the other two paths must be of the form $p_i \sim q_{ij} \sim q_{i+1,k} \sim p_{i+1}$. ■

It is fairly straightforward to generate by computer all plausible graphs Φ on 25 vertices that satisfy the properties in the lemma above. This results in 97 non-isomorphic graphs. As with the first algorithm we use these graphs to initialize some of the entries of a 25×25 distance matrix K. Apart from all entries with value 1, also some entries with value 2 and 3 can be initialized, in particular those corresponding to pairs of vertices $p, q \in \Phi$ with $d_\Phi(p,q) = 2$ and those corresponding to pairs of vertices $p_i \in P, q \in \Phi$ with $d_\Phi(p_i, q) = 3$.

The first step of the algorithm now recursively fills in the remaining entries of the distance matrix K with values 2 or 3 using the constraints of Section 4.1. In the second step of the algorithm we try to extend the generated matrices K to full 57×57 matrices M using the same constraints. Again we avoid to generate too many isomorphic results. This time we impose a lexical ordering on the columns in the 25×32 submatrix of M to the right of K. After approximately 15 million seconds of computer time (173 days) we obtained 33 versions of K that belong to only 2 isomorphism classes : in the first case $\Gamma(P)$ is the union of two disjoint pentagons and 10 isolated points, in the second case $\Gamma(P)$ is the disjoint union of 1 pentagon, 5 edges and 5 isolated points. In both cases, the union of P and a pentagon in its neighborhood induce a Petersen subgraph of Γ. In the second step of the algorithm both graphs Φ could be extended to a single graph (up to isomorphism) which in both cases turns out to be the 'true' Perkel graph. This last step took about 100 seconds.

It should be noted that the long running time for this second program is a little misleading and is largely a consequence of the fact that the algorithms used to check the constraints are different from the ones used in the first version. Where in the first program they were highly optimized for speed, we use more straightforward algorithms here whose correctness can be verified more easily although they are less efficient (e.g. preemptive positive semidefiniteness checking was omitted). We have used different algorithms on purpose, to increase the reliability of the results. Running the second program with the faster constraint algorithms we obtain the same results in approximately 320 000 seconds of computer time (4 days).

Bibliografie

[1] BAILEY R.A., *Association Schemes, Designed Experiments, Algebra and Combinatorics*, Cambridge studies in advanced mathematics, **84**, 2004.

[2] BANNAI E. AND ITO T., *Algebraic combinatorics I: Association schems*, Benjamin/Cummings, London, 1984.

[3] BATE J.A., HALL M. JR., VAN REES G.H.J., *Structures within* $(22, 33, 12, 8, 4)$*-designs*, J. Comb. Math. Combin Comput.,**4**, 1988, 115–122.

[4] BILIOUS R., LAM C.W.H., THIEL L.H., LI P.C., VAN REES G.H.J., RADZISZOWSKI S.P., HOLZMANN W.H, KHARAGHANI H., *There is no* $2-(22, 8, 4)$ *block design*, to appear in the Journal of Combinatorial Designs.

[5] BJÄRELAND M., JONSSON P., *Exploiting bipartiteness to identify yet another tractable subclass of CSP*, Lecture Notes in Computer Science, **1713**, 1999,118–128.

[6] BOSE R.C. AND MESNER D.M., *On linear associative algebras corresponding to association schemes of partially balanced designs*, Annals of Mathematical Statistics, **23**, 1959,21–38.

[7] BOSE R.C. AND SHIMAMOTO T., *Classification and analysis of partially balanced incomplete block designs with to associate classes*, Journal of the American Statistical Association, **47**, 1952,151–184.

[8] BRINKMANN G., *Isomorphism rejection in structure generation programs*, Discrete Mathematical Chemistry, DIMACS Ser. Discrete Math. Theoret. Comput. **51**, 2000, 25–38.

[9] Brinkmann G., *Fast Generation of Cubic Graphs*, J. Graph Th., **23**, 1996, 139–149.

[10] Brinkmann G., Dress A.W.M, *A constructive enumeration of fullerenes*, Journal of Algorithms, **23**, 1997, 345–358.

[11] Brinkmann G., McKay B.D., *Posets on up to 16 points*, Order, **19**, 2002, 147–179.

[12] Brouwer A. E., A. M. Cohen & A. Neumaier, *Distance-Regular Graphs*, Ergeb. Math. Grenzgeb. (3) **18**, Springer-Verlag, Berlin (1989).

[13] Brouwer A. E., Strongly Regular Graphs, in *The CRC handbook of Combinatorial Designs*, Chap. VI.5, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (1996).

[14] Brouwer A. E., Strongly Regular Graphs, in *The CRC handbook of Combinatorial Designs : second edition*, Chap. VII.11, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (2006).

[15] Brouwer A. E., Email server with information on distance-regular graphs, URL (2006): `www.win.tue.nl/~aeb/drginfo.txt`

[16] Bussemaker F.C., Haemers W.H., Mathon R., Wilbrink H.A., *A* $(49, 16, 3, 6)$ *strongly regular graph does not exist*, European Journal of Combinatorics, **10(5)**, 1989, 413–418.

[17] Butler G., *Fundamental Algorithms for Permutation Groups*, Lecture Notes in Computer Science **559**, Springer-Verlag, Berlin-New York, 1991.

[18] Caporossi G., Hansen P., *Enumeration of polyhex hydrocarbons to* $h =$ 21, Journal of Chemical Information and Computer Sciences, **38**, 1998, 610–619.

[19] Cauchie S. and Kuijken E. *Matrix techniques for strongly regular graphs and related geometries*, Supplement to the lecture notes for the Intensive Course on Finite Geometry and its Applications, Ghent, 2000.

[20] Coolsaet K., Fack V., *Classifying strongly regular graphs using lexical ordering of adjacency matrices*, Computers Math. Applic., **21**, No. 2-3, 1991, 15–21.

[21] Coolsaet K., Degraer J., *A computer assisted proof of the uniqueness of the Perkel graph*, Designs, Codes and Cryptography, **34**, 155-171, 2005.

[22] COOLSAET K., DEGRAER J., SPENCE E., *The strongly regular* $(45, 12, 3, 3)$ *graphs*, Electronic Journal of Combinatorics, Vol 13(1), R32, 2006.

[23] COOLSAET K., *The uniqueness of the strongly regular graph srg*$(105, 32, 4, 12)$, Bulletin of the Belgian Mathematical Society - Simon Stevin, Vol 12, No. 5, pp. 707-718, 2005.

[24] COOLSAET K., JURIŠIĆ A., *Using equality in the Krein conditions to prove nonexistence of certain distance regular graphs*, in preparation 2007.

[25] COLE F.N., CUMMINGS L.D., WHITE H.S., *The complete enumeration of triad systems of order 15*, Proceedings of the National Academy of Sciences of the United States of America 3, 1917, 197–199.

[26] COLBOURN C.J., DINITZ J. H. (EDS.), *The CRC handbook of Combinatorial Designs*, , CRC Press, Boca Raton (1996).

[27] COLBOURN C.J., Triple Systems, in *The CRC handbook of Combinatorial Designs : second edition*, Chap. II.2, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (2006).

[28] CHARTRAND C., LESNIAK L., *Graphs & Digraphs, third edition*, Chapman & Hall, 1996.

[29] DECHTER R., *Constraint Processing*, Morgan Kaufmann, 2003.

[30] DEGRAER J., COOLSAET K., *Classification of some strongly regular subgraphs of the McLaughlin graph*, Discrete Mathematics (to appear).

[31] DEGRAER J., COOLSAET K., *Classification of three-class association schemes using backtracking with dynamic variable ordering*, Discrete Mathematics, Vol 300, No. 1-3, pp. 71-81, 2005.

[32] DELSARTE P., *An algebraic approach to the association schemes of coding theory*, Philips Research Reports Suppl., **10**, 1973.

[33] DENNY P.C., GIBBONS P.B., *Case studies and new results in combinatorial enumeration,* J. Combin. Des.,**8(4)**, 2000, 239–260.

[34] DENNY P.C., MATHON R., *A census of* $t-(t+8, t+2, 4)$ *designs,* $2 \leq t \leq 4$, J. Statist. Plann. Inference, **106(1-2)**, 2002, 5–19.

[35] DENNY P.C., *Search and Enumeration Techniques for Incidence Structures*, CDMTS Research Report Series, University of Auckland,1998.

[36] DIESTEL R., *Graph Theory*, Graduate Texts in Mathematic, **173**, Springer-Verlag New York, Inc. , 1997.

[37] DINITZ J.H., GARNICK D.K., MCKAY B.D., *There are* $526, 915, 620$ *nonisomorphic* 1*-factorizations of* K_{12}, Journal of Combinatorial Designs,**2**, 1994, 273–285.

[38] DINITZ J.H., STINSON D.R., *A fast algorithm for finding strong starters*, SIAM J. Alg. Disc. Meth. **2**, (1981), 50–56.

[39] EMMS D. HANCOCK E.R., SEVERINI S., WILSON R.C., *A matrix representation of graphs and its spectrum as a graph invariant*, The Electronic Journal of Combinatorics, **13**, R34, 2006.

[40] FACK V., COOLSAET K., *An algorithm for the classification of stronly regular graphs by means of lexically ordered adjacency matrices.*, Intern. J. Computer Math. **33**, 1990, 143–151.

[41] FARADŽEV I.A., *Constructive enumeration of combinatorial objects*, Problèmes Combinatoires et Théorie des Graphes Colloque Internat. CNRS **260**, 1978, 131–135.

[42] FIALA N.C., HAEMERS, W.H., 5*-chromatic strongly regular graphs*, to appear in Discrete Mathematics, 2006.

[43] FON-DER-FLAASS D.G., *New prolific constructions of strongly regular graphs*, Adv.Geom, **2(3)**, 301–306, 2006.

[44] GASCHNIG J., *Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisfying assignement problems.* Proceedings of the Second Canadian Conference on Artifical Intelligence, 1978, 268–277.

[45] GIBBONS P.B, MATHON R., *The use of hill-climbing to construct orthogonal Steiner triple systems* Journal of Combinatorial Designs, **1**, 1993, 27–50.

[46] GOETHALS J.-M., J. J. SEIDEL, *The regular two graph on 276 vertices*, Discrete Mathematics, **12**, 1975, 143–158.

[47] GODSIL C.D., Association Schemes, in *The CRC handbook of Combinatorial Designs*, Chap. IV.1, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (1996).

[48] GODSIL C.D., *Problems in Algebraic Combinatorics*, Electronic Journal of Combinatorics, Vol 2, F1, 1995.

[49] GOLOMB S.W., BAUMERT L.D., *Backtrack programming*, Journal of the ACM, **12**, 1965, 516–524.

[50] GRUND R., KERBER A., LAUE R., *MOLGEN, ein Computeralgebra-System für die Konstruktion molekularer Graphen*, MATCH, **27**, 1992, 87–131.

[51] GRÜNER T., LAUE R., MERINGER M., *Algorithms for Group Actions: Homomorphism Principle and Orderly Generation Applied to Graphs*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **28**, 1997, 113–122.

[52] HAEMERS W.H., *Matrix techniques for strongly regular graphs*, Lecture notes for the Intensive Course on Finite Geometry and its Applications, Ghent, 2000.

[53] HAEMERS W.H., SPENCE E., *The Pseudo-Geometric Graphs for Generalised Quadrangles of Order* $(3,t)$, European J. Combin., **22(6)**, 2001, 839–845.

[54] HAEMERS W.H., TONCHEV V.D., *Spreads in strongly regular graphs*, Designs Codes Cryptography, **8**, 1996, 145–157.

[55] HAEMERS W. H., *Eigenvalue Techniques in Design and Graph Theory*, Mathematical Centre Tracts 121, Mathematisch Centrum, Amsterdam, (1980).

[56] HAEMERS W.H., KUIJKEN E., *The Hermitian two-graph and its code*, Linear Algebra and its Applications, **356(1–3)**, 2002, 79–93.

[57] HALL M. JR., ROTH R., VAN REES C.H.J., VANSTONE S.A., *On designs* $(22,33,12,8,4)$, J. Combin. Theory Ser. A,**47**, 1998, 157–175.

[58] HARALICK R., ELLIOTT G., *Increasing tree search efficiency for constraint-satisfaction problems*, Artificial Intelligence, **14(3)**, 1980, 263–313.

[59] HIGMAN D.G., *Coherent algebras*, Linear Algebra Appl., **93**, 1987, 109–239.

[60] HIRSCHFELD J.W.P., *Projective geometries over finite fields*, Oxford University Press, 1998.

[61] JØRGENSEN L.K., *Non-Symmetric 3-class association schemes*, preprint, submitted to Discrete Mathematics.

[62] JOOHYUNG K., *Classification of small class association schemes coming from certain combinatorial objects*, PHD dissertation, Iowa State University, 2006.

[63] KASKI P., ÖSTERGÅRD P.R.J, *The Steiner triple systems of order 19*, Mathematics of Computation, **73**, 2004, 2075–2092.

[64] KASKI P., ÖSTERGÅRD P.R.J, TOPOLOVA S., ZLATARSKI R., *Steiner triple systems of order 19 with subsystems of order 7*, Discrete Mathematics, to appear.

[65] KASKI P., ÖSTERGÅRD P.R.J, *Classification Algorithms for Codes and Designs*, Algorithms and Compuation in Mathematics, Springer, **15**, 2006.

[66] KOLESOVA G., LAM C.W.H., THIEL L., *On the number of* 8×8 *latin squares*, Journal of Combinatorial Theory A, **54**, 1990, 143-148.

[67] KUMAR V., *Algorithms for Constraint-Satisfaction problems: A survey*, AI Magazine, **13(1)**, 1992, 32–44.

[68] LAM C.W.H., THIEL L;, SWIERCZ S., *The nonexistence of finite projective planes of order* 10, Canad. J. Math., **45**, 1983, 319–321.

[69] MATHON R. AND SPENCE E., *On* $2-(45,12,3)$ *designs*, Journal of Combinatorial Design, **4(30)**, 1996, 155–175.

[70] MCKAY B.D., *Isomorph-free exhaustive generation*, Journal Algorithms **26(2)**, 1998, 306–324.

[71] MCKAY B.D., *Nauty users' guide (version 2.2)*, Technical report, Computer Science Department, Australian National University.

[72] MCKAY B., SPENCE E., *Classification of regular two-graphs on* 36 *and* 38 *vertices*, Australasian Journal of Combinatorics **24(1)**, 2001, 293–300.

[73] MCKAY B., RADZISZOWSKI S.P., *Towards deciding the existence of* $2-(22,8,4)$ *designs*, J. Combin. Math. Combin. Comput. **22**, 1996, 211–222.

[74] MCKAY B., MYRVOLD W., NADON J., *Fast backtracking principles applied to find new cages*, Proc. 9th ACM-SIAM Symposium on Discrete Algorithms , (San-Francisco, Jan 25–27, 1998), ACM Press, New York, 1998, 188–191.

[75] MCKAY B.D., MEYNERT A., MYRVOLD W., *Small Latin squares, quasigroups and loops*, Journal Combinatorial Designs, to appear.

[76] MCKAY B.D., *personal communication*, Symnet Summer School, St Andrews, 2004.

[77] MCKAY B. D., *Practical Graph Isomorphism*, Congressus Numerantium, **30** ,1981,45–87.

[78] MCKAY B.D., *Combinatorial Data*, `http://cs.anu.edu.au/~bdm/data/graphs.html`, (2006).

[79] MERINGER M., *Fast Generation of Regular Graphs and Construction of Cages*, Journal of Graph Theory **30**, 1999, 137–146.

[80] MERINGER M., *Erzeugung regulärer Graphen* Master's thesis, Universität Bayreuth, January 1996.

[81] NURMELA K.J.,ÖSTERGÅRD P.R.J., *Upper bounds for covering designs designs by simulated annealing*, Congr. Numer., 96, 1993, 93–111.

[82] ÖSTERGÅRD P.R.J., *Enumeration of* $2-(12,3,2)$ *designs*, Australas. J. Combin., **22**, 2000, 227–231

[83] ÖSTERGÅRD P.R.J., KASKI P., *Enumeration of* $2-(9,3,\lambda)$ *designs and their resolutions*, Designs Codes and Cryptography, **27**, 2002, 131-137.

[84] ÖSTERGÅRD P.R.J., *New multiple covering codes by tabu search*, The Australasian Journal of Combinatorics, **12**, 145–155.

[85] ÖSTERGÅRD P.R.J., *A* $2-(22,8,4)$ *design cannot have a* $2-(10,4,4)$ *subdesign*, Designs Codes and Cryptography, **27**, 2002, 257–260.

[86] PAULUS A.J.L., *Conference matrices and graphs of order* 26, Technische Hogeschool Eindhoven, report WSK 73/06, Eindhoven, 1983.

[87] READ R.C., *Every one a winner or How to avoid isomorphism search when cataloguing combinatorial configurations*, Ann. Discrete Math. **2**, 1978, 107–120.

[88] ROYLE G., *Cubic Graphs*, `http://people.csse.uwa.edu.au/gordon/remote/cubics/index.html`, (2006).

[89] SCHMALZ B., *The t-designs with prescribed automorphism group*, Journal Combinatorial Design, **1**, 1993, 125–170.

[90] SEAH E., STINSON D.R., *A perfect* 1*-factorisation of* K_{40}, Congr. Numer., 68, 1989, 211-214.

[91] SEDGEWICK R., *Algorithms in C++: Parts 1–4*, Addisson–Wesley, 1998.

[92] SEDGEWICK R., FLAJOLET P., *An introduction to the analysis of algorithms*, Addisson–Wesley, 1996.

[93] SEIDEL R., SHARIR M., *Top-down analysis of path compression*, SIAM J. Comput. **34(3)**, 2005, 515-525

[94] SERESS A., *Permutation group algorithms*, Cambridge Tracts in Mathematics, Cambridge University Press **152**, 2003.

[95] SPENCE E., *The Strongly Regular* $(40,12,2,4)$ *graphs*, Electronic Journal of Combinatorics, **7(1)**, 2000.

[96] SPENCE E., *Regular two-graphs on 36 vertices*, Linear Algebra and its Applications, **226-228**, 1995, 459–497.

[97] SPENCE E., *Strongly Regular Graphs*, `http://www.maths.gla.ac.uk/~es/srgraphs.html`, (2006).

[98] SPENCE E., $(40, 13, 4)$ *designs derived from strongly regular graphs*, Advances in Finite Geometry and Designs, Oxford University Press, 1990, 359–368.

[99] SPENCE E., *Regular Two-Graphs*, in The CRC handbook of Combinatorial Designs : second edition, Chap. VII.13, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (2006).

[100] SMITH D.H., *Primitive and imprimitive graphs*, Quart. J. Math. Oxford, **22(2)**, 1971, 551-557.

[101] VAN DAM E. *Three-class association schemes*, Journal of Algebraic Combinatorics, **10(1)**, 1999,69-107(39)

[102] VAN DAM E., HAEMERS W. H., KOOLEN J. H., SPENCE E., *Characterizing Distance Regularity of Graphs by the Spectrum*, to appear in J. Combin. Theory, Series A.

[103] VAN DAM E., HAEMERS W. H., *Which graphs are determined by their spectrum?* Linear Algebra and its Applications, **373**, 2003, 241-272.

[104] VAN LEEUWEN J., TARJAN R.E., *Worst-case analysis of set-union algorithms*, Journal of the ACM, 1984.

[105] WEST D.B., *Introduction to Graph Theory*, Prentice Hall, 1996.

[106] WINNE J., *Software tools for combinatorial algorithms*, Phd dissertation, Ghent University, 2007.

[107] WILSON R.M., *Nonisomorphic Steiner triple systems*, Math. Z.,**135**, 1974, 303–313, MR 49:4803

Nederlandse samenvatting

Combinatoriek is een tak van de wiskunde. In de combinatoriek bestudeert men eindige verzamelingen van objecten die aan bepaalde eigenschappen voldoen. In het bijzonder houdt men zich bezig met volgende problemen:

- Bestaat er een object dat aan deze eigenschappen voldoet?
- Gegeven een verzameling eigenschappen, tel het aantal objecten die voldoen aan deze eigenschappen.
- Gegeven een verzameling eigenschappen, classificeer alle objecten die voldoen aan deze eigenschappen of toon aan dat deze objecten niet bestaan.

Voor tel- en classificatieproblemen is het gebruikelijk te veronderstellen dat de objecten onderhevig zijn aan een relatie 'is isomorf met' die deze objecten partitioneert in equivalentieklassen. Zodoende wordt slechts één representatieve van elke equivalentieklasse geteld of geclassificeerd. Objecten in eenzelfde equivalentieklasse zijn isomorf en worden wiskundig als identiek beschouwd. In het algemeen zijn twee objecten isomorf als het ene object kan bekomen worden uit het andere – en omgekeerd – door het hernoemen en/of herlabelen van meer elementaire objecten waaruit deze objecten bestaan.

Algoritmen zijn een essentieel middel geworden in het succesvol oplossen van zowel bestaans- als classificatieproblemen. Combinatorische objecten zijn zowel eindige als discrete structuren wat maakt dat dergelijke problemen kunnen aangepakt worden met lokale en exhaustieve zoekalgoritmen. Enerzijds worden lokale zoekalgoritmen gebruikt om bestaansproblemen op te lossen. Deze methoden blijken vaak efficiënt te zijn alhoewel ze niet garanderen dat een oplossing gevonden wordt, zelfs wanneer één of meerdere oplossingen bestaan. Exhaustieve zoekalgoritmen anderzijds, worden vaak gebruikt om classificatieproblemen op te lossen. Deze methoden overlopen systematisch alle kandidaatoplossingen binnen een gegeven zoekruimte en garanderen dus alle oplossingen te vinden, indien die bestaan.

Wellicht de meest gekende exhaustieve zoekmethode is *backtracking*, waarbij men een oplossing probeert te bekomen door telkens een deeloplossing recursief uit te bereiden in de richting van de uiteindelijke oplossing. Deze uitbereiding wordt later ongedaan gemaakt als blijkt dat die niet tot een oplossing leidt. Backtracking kan gezien worden als een diepte-eerst overlopen van een zoekboom waarin alle knopen overeenkomen met deeloplossingen en waarbij takken in de zoekboom overeenkomen met het systematisch uitbereiden van een deeloplossing, of in omgekeerde richting, met het systematisch ongedaan maken van deze uitbereiding.

In dit doctoraat beschouwen we algoritmen voor het classificeren van combinatorische objecten op isomorfie na, of zogenaamde isomorfvrije exhaustieve generatiealgoritmen. We concentreren ons op het isomorfvrij genereren van associatieschema's [1, 47] en sterk reguliere [13, 14] en afstands reguliere [12] grafen in het bijzonder. De nadruk ligt op het ontwerpen en verbeteren van isomorfvrije exhaustieve generatiealgoritmen die uiteindelijk vlug genoeg blijken om nieuwe classificatieresultaten te bekomen.

Dit doctoraatsproefschrift is opgebouwd als volgt. In Hoofdstuk 2 geven we een overzicht van de theoretische achtergrond omtrent associatieschema's, sterk reguliere en afstands reguliere grafen. Een d-klasse associatieschema Ω met toppenverzameling $V = \{1, \ldots, v\}$ bestaat uit een verzameling van $d + 1$ symmetrische relaties $\{R_0, \ldots, R_d\}$ op V, met identiteitsrelatie R_0 zodanig dat twee toppen deeluitmaken van slechts één enkele relatie. Bovendien, bestaan er intersectie getallen p^k_{ij} zodanig dat voor elke $(x, y) \in R_k$, het aantal toppen z zodanig dat $(x, z) \in R_i$ en $(z, y) \in R_j$ gelijk is aan p^k_{ij}. Als de unie van sommige relaties een niet-triviale equivalentierelatie vormt, dan noemt men Ω imprimitief – anders noemt men het primitief. Een associatieschema Ω kan voorgesteld worden a.d.h.v. zijn $v \times v$ symmetrische relatiematrix M_Ω waarbij een matrixpositie $(M_\Omega)_{xy} = i$ als en slechts als $(x, y) \in R_i$.

Een afstands reguliere graaf is een geconnecteerde graaf waarvan de afstandsrelaties een associatieschema vormen. Een sterk regulier graaf is een afstands reguliere graaf met diameter 2. Een geconnecteerde graaf is sterk regulier met parameters (v, k, λ, μ) als en slechts als deze graaf v toppen heeft, regulier is van graad k, elk twee adjacente toppen precies λ gemeenschappelijke buren hebben en elke twee niet-adjacente toppen precies μ gemeenschappelijke buren hebben

De niet-triviale relaties R_i van een associatieschema kunnen beschouwd worden als regulier grafen van graad p^0_{ii}. Voor elke corresponderende adjacentiematrix A_i zijn de axioma's van een d-klasse associatieschema equivalent met

$$\sum_{l=0}^{d} A_l = J, \quad A_0 = I, \quad A_i = A_i^T \quad \text{en} \quad A_i A_j = \sum_{l=0}^{d} p^l_{ij} A_l.$$

Hieruit volgt dat de adjacentiematrices $A_0, \ldots, A_d$ een $(d+1)$-dimensionale commutatieve algebra $\mathcal{A}$ genereren. Deze algebra $\mathcal{A}$ wordt de Bose-Mesner algebra van het associatieschema genoemd. De algebra $\mathcal{A}$ heeft een unieke basis van minimaal idempotenten E_i zodanig dat

$$E_i E_j = \begin{cases} E_i, & \text{als } i = j \\ 0, & \text{als } i \neq j \end{cases} \quad \text{en} \quad \sum_{i=0}^{d} E_i = I$$

Elke minimaal idempotente kan als volgt worden uitgedrukt in termen van adjacentiematrices (en omgekeerd),

$$v\,E_i = \sum_{j=0}^{d} Q_{ji} A_j \quad \text{en} \quad A_j = \sum_{i=0}^{d} P_{ij} E_i.$$

Hieruit volgt dat $PQ = QP = v\,I$ en $A_j E_i = P_{ij} E_i$ waarbij P_{ij} een eigenwaarde is van A_j met multipliciteit $m_i = \text{rang}(E_i)$. De $(d+1) \times (d+1)$ matrices P en Q worden de eigenmatrices van het associatieschema genoemd. De $(d+1) \times (d+1)$ intersectiematrices L_i waarbij $(L_i)_{kj} = p^k_{ij}$ hebben dezelfde eigenwaarden P_{ji} waarbij de eigenvectoren gegeven zijn door de kolommen van Q.

In Hoofdstuk 3 beschrijven we enkele algemene technieken die gebruikt worden in het ontwikkelen van isomorfvrije generatiealgoritmen. Zulke algoritmen vereisen een recursief doorlopen van een boomachtige zoekruimte. Het is gebruikelijk om verschillende intelligente snoeimethoden toe te passen om de grootte van deze exponentiële zoekruimte te beperken. Enerzijds kunnen we takken van de zoekboom snoeien die overeenkomen met deeloplossingen waarvan aangetoond kan worden dat ze onmogelijk uit te bereiden zijn tot een oplossing die aan de gespecificeerde eigenschappen voldoet. Hierbij wordt meestal gesteund op wiskundige eigenschappen van de objecten die men wil genereren. Deze eerste soort snoeitechnieken wordt geïntroduceerd binnen het algemene kader van 'constraint'-netwerken [29].

Anderzijds kunnen kunnen we takken snoeien die overeenkomen met deeloplossingen waarvan aangetoond kan worden dat ze isomorf zijn met deeloplossingen die we reeds in de zoekboom tegengekomen zijn. Merk ook op dat deze laatste snoeimethoden er voor moeten zorgen dat slechts één representatieve van elke isomorfieklasse gegenereerd wordt. We leggen de nadruk op 'orderly'-algoritmen [87]. In deze aanpak is er één canonisch gelabeld object in elke isomorfieklasse en dat is de representatieve die gegenereerd wordt. Hierbij is het gebruikelijk om een canonische vorm te kiezen zodat specifieke deelobjecten van canonisch gelabelde objecten ook in canonische vorm zijn. Deze tweede soort snoeitechnieken wordt geïntroduceerd binnen een algemeen groeptheoretisch kader.

Specifieke isomorfvrije exhaustieve generatiealgoritmen voor d-klasse associatieschema's die voldoen aan een gegegeven verzameling intersectiematrices $\{L_0, \ldots, L_d\}$ worden besproken in Hoofdstuk 4 en 5. Deze generatiealgoritmen starten initieel met een relatiematrix M waarbij enkel de diagonaalposities geïnstantieerd zijn. Elke matrixpositie M_{xy} met $x < y$ – en op hetzelfde moment ook zijn symmetrisch tegenhanger M_{yx} – wordt systematisch recursief geïnstantieerd met elk waarde van het domein $\{1, \ldots, d\}$. Knopen op diepte k in de zoekboom stellen gedeeltelijk geïnstantieerde relatiematrices M voor waarbij exact k matrixposities boven de diagonaal ingevuld zijn.

Aan de hand van 'constraint'-netwerken worden specifieke snoeitechnieken voor associatieschema's beschreven. Enerzijds zijn deze snoeitechnieken gebaseerd op combinatorische eigenschappen bepaald door de definiërende axioma's van een asscociatieschema. E.g., een rij van een gedeeltelijk geïnstantieerde relatiematrix M kan hoogstens p^0_{kk} matrixposities met de waarde k bevatten. Anderzijds worden ook algebraïsche eigenschappen van de Bose-Mesner algebra gebruikt als snoeicriterium. Elke minimaal idempotente E_i heeft een rang Q_{0i}

en is positief semidefiniet. Een belangrijk kenmerk van deze algebraïsche eigenschappen is dat elke principale deelmatrix van E_i positief semidefiniet moet zijn en een rang hebben die hoogsten Q_{0i} bedraagt.

In een groeptheoretisch kader worden rijgeordende en kolomgeordende canonische vormen gedefinieerd. Beide canonische vormen hebben gemeen dat ze minimaal zijn in hun isomorfieklasse, afhankelijk van een welbepaalde lexicografische ordening. Meer bepaald, de rijgeordende canonische vorm van een relatiematrix M is gedefinieerd als de lexicografisch kleinste string die bekomen kan worden door de rijen en kolommen van M te permuteren waarbij de rijen van het gedeelte van M boven de diagonaal geconcateneerd worden. Bij een kolomgeordende canonische vorm daarentegen bekomt men de lexicografisch kleinste string door de kolommen van het gedeelte van M boven de diagonaal te concateneren. Merk op dat het gebruik van een rijgeorderende en een kolomgeordende canonische vorm vereist dat de matrixposities van M respectievelijk rij per rij en kolom per kolom geïnstantieerd worden. Als slot van Hoofdstuk 4 wordt de interactie van beide canonische vormen met de combinatorische en algebraïsche snoeicriteria geëvalueerd. Hieruit blijkt dat het gebruik van een kolomgeordende canonisch vorm beter presteert. De bijhorende instantiatieorde van M zorgt er voor dat grote principale submatrices van de minimaal idempotenten E_i vlugger bekomen worden tijdens het recursieve generatieproces.

In Hoofdstuk 5 ontwikkelen we een canoniciteitsalgoritme dat bepaald of een gegeven matrix M in kolomgeordende canonisch vorm is. De enige gekende aanpak is om via bactracking alle mogelijke permutaties van de rijen en kolommen van M te beshouwen. Verschillende snoeitechnieken zijn mogelijk om de corresponderende recursieboom te snoeien. Niettemin, blijft het ontwikkelen van een efficiënt canoniciteitsalgoritme een gecompliceerde en subtiele opdracht. De ontwikkeling van het algoritmen wordt beschreven in opeenvolgende stappen. Bij elke stap worden de gemaakte ontwerpsbeslissingen en hun impact geïllustreerd aan de hand van empirische data. In een eerste fase wordt het canoniciteitsalgoritmen ontwikkeld enkel rekening houdend met het feit dat het algoritme een symmetrische matrix M als input krijgt. In een tweede fase zijn de bijkomende verbeteringen gebaseerd op het feit dat het canoniciteitsalgoritme herhaaldelijk gebruikt wordt gedurende de loop van een ‘orderly-generatiealgoritme.

In Hoofdstuk 6 tenslotte, geven we een overzicht van verschillende nieuwe classifcatieresultaten voor associatieschema's, sterke regulier en afstands regulier grafen bekomen met de generatiealgoritmen beschreven in Hoofdstuk 4 en 5. Sommige van deze classificatieresultaten werden reeds gepubliceerd in gespecialiseerde wetenschappelijke tijdschriften [21, 22, 30, 31].