The London School of Economics and Political Science

Integer and Constraint Programming methods for Mutually Orthogonal Latin Squares

PhD Thesis

by

Ioannis Mourtos

London, 2003

Abstract

This thesis examines the *Orthogonal Latin Squares* (OLS) problem from the viewpoint of Integer and Constraint programming. An *Integer Programming* (IP) model is proposed and the associated polytope is analysed. We identify several families of strong valid inequalities, namely inequalities arising from cliques, odd holes, antiwebs and wheels of the associated intersection graph. The dimension of the OLS polytope is established and it is proved that certain valid inequalities are facet-inducing. This analysis reveals also a new family of facet-defining inequalities for the polytope associated with the Latin square problem. Separation algorithms of the lowest complexity are presented for particular families of valid inequalities.

We illustrate a method for reducing problem's symmetry, which extends previously known results. This allows us to devise an alternative proof for the non-existence of an OLS structure for $n = 6$, based solely on Linear Programming. Moreover, we present a more general *Branch & Cut* algorithm for the OLS problem. The algorithm exploits problem structure via integer preprocessing and a specialised branching mechanism. It also incorporates families of strong valid inequalities. Computational analysis is conducted in order to illustrate the significant improvements over simple *Branch & Bound.*

Next, the Constraint Programming (CP) paradigm is examined. Important aspects of designing an efficient CP solver, such as branching strategies and constraint propagation procedures, are evaluated by comprehensive, problem-specific, experiments. The CP algorithms lead to computationally favourable results. In particular, the infeasible case of $n = 6$, which requires enumerating the entire solution space, is solved in a few seconds.

A broader aim of our research is to successfully integrate IP and CP. Hence, we present ideas concerning the unification of IP and CP methods in the form of hybrid algorithms. Two such algorithms are presented and their behaviour is analysed via experimentation. The main finding is that hybrid algorithms are clearly more efficient, as problem size grows, and exhibit a more robust performance than traditional IP and CP algorithms. A hybrid algorithm is also designed for the problem of finding triples of *Mutually Orthogonal Latin Squares* (MOLS).

Given that the OLS problem is a special form of an assignment problem, the last part of the thesis considers multidimensional assignment problems. It introduces a model encompassing all assignment structures, including the case of MOLS. A necessary condition for the existence of an assignment structure is revealed. Relations among assignment problems are also examined, leading to a proposed hierarchy. Further, the polyhedral analysis presented unifies and generalises previous results.

Acknowledgments

I would like to thank Dr. Gautam M. Appa for being a creative and patient supervisor for the last three years. Apart from being a source of interesting ideas, Dr. Appa has provided vital personal support. Most important, his example of an intellectual has had strong influence on my perception of academic attitude.

The Operational Research Department of the London School of Economics has been an excellent academic environment for conducting research and a place full of affable people. My special thanks to Brenda Mowlam, whose support and sense of humour always made things look much simpler. I should also mention Dr. Susan Powell's motivation in starting my PhD and her continuing support thereafter. I feel also fortunate for having had as fellow PhD students my friends Balazs Kotnyek and Monica Oliveira.

Dr. Dimitris Magos has been a valuable colleague, who soon became a close friend. Working with him has been equally stimulating and enjoyable. I would also like to thank Dr. Hugues Marchand, Dr. Jeanette Janssen, Professor A.D. Keedwell and Professor H.P. Williams for their advisory role in various stages of my study.

This research has been accomplished under scholarhips awarded by the Bodossaki Foundation and the Institute of State Scholarships in Greece. I should also acknowledge the financial support of the London School of Economics and the Discrete Optimization Network of the European Union.

Preface

This thesis considers the problem of Mutually Orthogonal Latin Squares (MOLS) from the perspectives of Integer Programming (IP) and Constraint Programming (CP). In Chapter 1, we present basic definitions and facts from the theory of Latin squares, emphasising the results related to our research. A number of related fields and applications are discussed in order to illustrate the problem's importance and to motivate the following analysis. We also devise and compare two IP formulations for the problem of Orthogonal Latin Squares (OLS). Finally, we provide a description of the convex-hull of a relaxed problem for $n = 2$, in the form of families of valid inequalities. This final section anticipates the forthcoming analysis.

The convex hull of integer vectors, which are feasible with respect to the IP model, defines the OLS polytope. Integer points in this polytope have a one-to-one correspondence to OLS structures of order n. Moreover, it is shown that the OLS problem is equivalent to the planar 4-index assignment problem. To sufficiently characterise this combinatorial object, we need to obtain families of, polyhedrally strong, valid inequalities. This is the aim of Chapter 2. The main method for obtaining valid inequalities is the analysis of the intersection graph associated with the IP model. We identify certain subgraphs of the intersection graph, study the form of induced inequalities and strengthen these inequalities via sequential lifting. Specifically, we describe and count all clique inequalities, all inequalities arising from wheels of a particular type, a family of antiweb inequalities, a family of composite cliques and two families of odd-hole inequalities. We also prove that no odd anti-hole subgraphs exist.

Chapter 3 applies the tools of polyhedral combinatorics to the OLS polytope. This polytope exhibits certain irregularities, which can be handled only by exploiting the problem's structure. For example, it is difficult to illustrate a trivial integer vector, i.e. an OLS structure, for every value of n. After identifying the rank of the constraint matrix, we are able to establish the dimension of the OLS polytope. The next step is to identify facets of this polytope. We show which of the linear constraints defining the problem's LP-relaxation give rise to trivial facets. Most important, we examine the polyhedral properties of the inequalities presented in Chapter 2, in terms of their dimension and their Chvâtal rank. It is shown that two families of clique inequalities and the family of composite clique inequalities induce facets of the OLS polytope. In this context, we also identify a new family of facets for the polytope associated with the Latin square problem. Further, we provide separation algorithms for clique, lifted antiweb and lifted odd-hole inequalities. These algorithms determine whether a given fractional LP-solution violates any of these inequalities. Their complexity is linear in the number of variables.

Separation procedures are a prerequisite for the design of a Branch & Cut algorithm. This is precisely the topic addressed in Chapter 4. At this point, our focus is transferred to algorithmic aspects and remains as such until Chapter 7. We should note that the OLS problem is basically treated as a feasibility problem. Hence, Chapter 4 begins with an alternative proof, solely based on

Linear Programming, for the non-existence of OLS for $n = 6$. This instance was the first studied by L. Euler in the 18th century. To achieve this result, we propose a method for eliminating symmetries. Afterwards, we present the components of our IP algorithm, namely a problem-specific preprocessor, a specialised branching rule based on Special Ordered Sets and a cut generator, which utilises the results of Chapter 3. Computational results reveal the extent to which each of these components improves over a Branch & Bound algorithm. Overall, the Branch & Cut algorithm accelerates the solution time approximately by a factor of 2.

The Constraint Programming approach is illustrated in Chapter 5. After providing a logic proof for the infeasibility of OLS for $n = 2$, we review the related literature. Next, we design a CP algorithm for the OLS problem. This algorithm incorporates various procedures for constraint propagation within a Forward Checking algorithmic framework. Essentially, it provides an enumerative approach analogous to our IP algorithm, without solving the LP-relaxation. Problem-specific features allow this approach to produce favourable computational results and, in particular, to prove infeasibility for $n = 6$ much more rapidly.

Chapter 6 is the culmination of the previous two chapters in terms of comparing and integrating IP and CP methods. First, we discuss theoretical links between logic and optimisation, along with algorithmic ideas concerning the integration of IP and CP. Afterwards, we implement these ideas in the form of two hybrid algorithm for the OLS problem. The computational analysis shows that hybrid methods exhibit a more balanced performance. Especially one hybrid algorithm performs better for higher orders, a fact that allows more general conclusions to be drawn. The final sections of this chapter present algorithms for identifying a triple of MOLS. The algorithm solves the problem efficiently for up to $n = 9$ but returns no solution for $n = 10$. The existence of an MOLS triple for $n = 10$ is probably the most famous unsolved instance of MOLS. We show that our hybrid algorithm is more efficient, compared to traditional IP and CP methods. We also provide an estimate of the time required to enumerate the entire solution space, in case the problem is infeasible.

It has been said that the inclination of lawyers towards prosecuting is exceeded only by the propensity of mathematicians to generalise. Hence, Chapter 7 is mainly devoted to generalising our polyhedral analysis to multidimensional assignment problems. We introduce an IP model, provide a necessary condition for the existence of an integer feasible vector and propose a hierarchy among assignment problems. Establishing the dimension of classes of assignment polytopes unifies and generalises previously known results. Apart from identifying trivial facets, we also prove that a certain family of clique inequalities induces facets of the axial assignment polytope. The last section introduces CP models and discusses methods for extending the hybrid algorithms of Chapter 6 to multidimensional assignment problems.

Contents

Chapter 1

The Problem of Mutually Orthogonal Latin Squares

It is natural for a thesis focused on Mutually Orthogonal Latin Squares to initiate the discussion by presenting the problem itself. Latin Squares and their extensions define an entire field of finite algebra, having cross-links to multiple areas of mathematics. Therefore, the related literature is enormous. An extensive discussion of the subject, involving old and more recent developments can be found in [34, 35, 60]. We restrict ourselves to review only the aspects, which are relevant to this work and act as a motivation for the following chapters.

Section 1.1 provides a number of introductory definitions, which will be used throughout the thesis. Important facts concerning the theory of Latin squares are presented in Section 1.2. Section 1.3 reveals the strong connection of Latin squares to certain areas of discrete mathematics, while Section 1.4 presents a series of applications. In Section 1.5, we provide and compare two IP formulations for the OLS problem. Finally, Section 1.6 demonstrates the convex hull of all feasible solutions to the OLS problem for $n = 2$, thus anticipating the developments of forthcoming chapters.

1.1 Definitions

The following definitions are obtained from [34], unless otherwise stated.

Definition 1.1 *A* Latin *square of* order n *is a square matrix of order* n*, where each value* $0, .., (n-1)$ *appears exactly once in each row and column.*

Consider, as an example, the squares of Table 1.1.

Table 1.1: A pair of *OLS* of order 4

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

L_1

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

L_2

Latin squares can also be defined as multiplication tables (often called the *Cayley* tables) of *quasigroups.* Before defining quasigroups, let us recall the definitions of a group.

Definition 1.2 *A finite group, denoted as a tuple $(S, *)$, is defined as a non-empty finite set S together with a binary operation $(*)$, having the following properties:*

(i) $a * (b * c) = (a * b) * c$, for all $a, b, c \in S$, i.e. operation $*$ is *associative*;

(ii) there is an *identity element* $e \in S$, such that $a * e = e * a = a$, for all $a \in S$;

(iii) for each $a \in S$, there is an *inverse element* $a^{-1} \in S$, such that $a * a^{-1} = a^{-1} * a = e$.

An implicit property is that, for all $a, b \in S$, it holds that $a * b \in S$. As an example, consider the group $(Z_n, +)$, where $Z_n = \{0, ..., n-1\}$ and operator '+' stands for addition *modulo n*. The multiplication table of a group $(S, *)$ is a table, whose rows and columns are labelled with the elements of S and cell (a, b) contains value $a * b$. Table 1.2 illustrates the multiplication table of the group $(Z_4, +)$.

Table 1.2: The multiplication table of $(Z_4, +)$

	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Hence, the multiplication table of a group is a Latin square. However, not all Latin squares are multiplication tables of groups (see [60, Section 6.2]). It is therefore reasonable to ask whether there exists an algebraic structure such that its multiplication table is a Latin square and, conversely, every Latin square represents the multiplication table of such a structure. The answer exists in the class of quasigroups, which encompasses that of groups.

Definition 1.3 *[34, p.16] A set S, on which a binary operation $(\cdot)$ is defined, forms a quasigroup with respect to that operation if, for any $a, b \in S$, it holds that $a \cdot b \in S$ and each of the equations $a \cdot x = b$ and $y \cdot a = b$ has exactly one solution in S.*

A special category of Latin Squares is the following.

Definition 1.4 *A Latin square is said to be in reduced or standard form if the elements $0, ..., (n-1)$ occur in the first row and column in their natural order.*

One can permute the n columns of a reduced Latin squares in $n!$ ways. It is easy to see that the resulting squares remains Latin and are all distinct. Clearly, none of them is reduced. At each of these squares, one can also permute the last $(n-1)$ rows in $(n-1)!$ possible ways, again forming an equal number of distinct Latin squares. Let L_n, l_n denote the number of Latin squares and reduced Latin squares of order n, respectively. We have essentially proved the following.

Theorem 1.1 *For each $n \geq 2$, the number L_n of Latin squares of order n is*

$$L_n = n!(n-1)!l_n$$

A more formal proof can be found in [60, Section 1.2]. Let us now define the concept of Orthogonal Latin Squares in terms of pairs of elements appearing in the n^2 cells of Latin squares L_1 and L_2.

Definition 1.5 *Two Latin squares of order n are called* orthogonal (OLS) *if and only if each of the* n^2 *ordered pairs* $(0,0),...,(n-1,n-1)$ *appears exactly once in the two squares.*

A pair of *OLS* of order 4 appears in Table 1.1. This definition is naturally extended to sets of $k > 2$ Latin squares, which are called *Mutually Orthogonal (MOLS)* if they are pairwise orthogonal. Extending the idea of a reduced Latin square, we call a set of k MOLS *reduced* or *standardised* if elements $0,..,(n-1)$ appear at the first row of all squares and at the first column of exactly one square in natural order. A concept related to OLS is that of a *transversal.*

Definition 1.6 *A transversal of a Latin square is a set of n cells, each in a different row and column, which contain n pairwise different values.*

Denote the squares of Table 1.1 as L_1 and L_2. For each set of 4 cells in square L_2 that contain the same value, observe that the corresponding cells of square L_1 form a transversal. An easy to prove but quite significant property of OLS is the following.

Theorem 1.2 *[34, Theorem 5.1.1] A Latin square of order n has an orthogonal mate if and only if it can be decomposed into n disjoint transversals.*

OLS were introduced by Euler via his famous problem of the 36 army officers [36]. These officers should belong to 6 distinct ranks, 6 to each one, and are collected from 6 different regiments (6 officers from each regiment). The 36 officers can be arranged into a 6 × 6 grid such that exactly one officer of each rank appears in each row and column. By definition, this is a Latin square L_1. Similarly, the officers can be arranged into another square L_2 according to their regiment, i.e. exactly one officer from each regiment appears in each row and column. The number of reduced Latin squares of order 6 is known to be 9408. Hence, these arrangements can be devised in $6!5! \times 9408 \simeq 10^9$ ways, which is the number of Latin squares of order 6. The question is whether there exist squares L_1 and L_2 such that all 36 rank-regiment combinations appear exactly once, i.e. whether there is a pair of OLS of order 6. Euler's belief about the non-existence of such a pair was formally proved by Tarry ([81]), essentially by exhaustive enumeration. Shorter proofs have also been proposed, for example in [80].

It is trivial to show that there can exist no pair of OLS for $n = 2$. Motivated by the non-existence of OLS for $n = 2, 6$, Euler stated a conjecture that no pair of OLS exists if n is an odd multiple of 2. The falsity of this conjecture was shown in 1959 ([19]) by constructing a pair of OLS for $n = 22$. This celebrated result revived the interest in Orthogonal Latin squares and their properties.

1.2 Theory of Latin Squares

In this section, we state certain theorems regarding Latin squares and OLS, without their proofs. The related theory is much broader and is still being augmented. Hence, only results relevant to our research are illustrated.

1.2.1 Single Latin Squares

Recall the simple observation that a square remains Latin after permuting its columns. In fact, a Latin square embodies three entities, conventionally corresponding to the n-sets of rows, columns and values. Apparently, any of these sets can be permuted without violating the fact that each value must exist exactly once in each row and column. The following statement provides a formal definition in terms of quasigroups.

Definition 1.7 *Let $(G,\cdot)$ and $(H,*)$ be two quasigroups. An ordered triple (θ,ϕ,ψ) of $1-1$ mappings θ,ϕ,ψ of the set G onto the set H is called an* isotopism *of $(G,\cdot)$ upon $(H,*)$ if $(x\theta)*(y\phi)=(x\cdot y)\psi$ for all $x,y\in G$. The quasigroups are then said to be* isotopic. *If $\theta=\phi=\psi$ such a transformation is called an* isomorphism.

The definitions of isotopism and isomorphism are obtained from [34]. To avoid confusion, we have to remark that numerous textbooks in graph theory and combinatorics (e.g. [17]) assign to isomorphism the properties of what is defined here as isotopism. Clearly, two Latin squares are called *isotopic* if they define the multiplication tables of isotopic quasigroups. As an example, consider square L_1 of Table 1.1. In order to derive an isotopic square $\hat{L}_1$, we need to define three different permutations for the sets of rows, columns and elements of L_1. If we apply the permutations $\theta=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$, $\phi=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$ and $\psi=\begin{pmatrix}0&1&2&3\\0&1&2&3\end{pmatrix}$ to rows, columns and elements of L_1, we obtain the isotopic square $\hat{L}_1$ of Table 1.3. For example, notice that cell $(2,3)$ of L_1, containing value 1, is mapped to $(1,2)$ in $\hat{L}_1$, also containing value 1. Similarly, to derive the square $\tilde{L}_1$ of Table 1.3, which is isomorphic to L_1, we apply the same permutation $\theta=\phi=\psi=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$ to rows, columns and elements of L_1.

Table 1.3: Isotopic and isomorphic squares of L_1

3	1	0	2
2	0	1	3
0	2	3	1
1	3	2	0

$\hat{L}_1$

2	3	0	1
3	2	1	0
0	1	2	3
1	0	3	2

$\tilde{L}_1$

Hence, isotopy is a property encompassing the inherent symmetry of Latin squares. A less obvious form of symmetry is obtained if we observe that the roles of rows, columns and values are interchangeable. For example, by interchanging the roles of rows and columns of a Latin square L, we derive its transpose L^T, which is also a Latin square. Since there exist 3! permutations of the three entities, each Latin square may be used to obtain up to 5 other Latin squares L'. Any such square L' is said to be *conjugate* to L. Notice that the conjugate squares of a Latin square L are not necessarily distinct.

Theorem 1.3 *[34, Section 1.4] The number of distinct conjugates of a Latin square L is always 1,2,3 or 6.*

Isotopy and isomorphy are associative and commutative properties. In this sense, they define equivalence relations among Latin squares of the same order. Let Ω_n denote the set of Latin squares of order n. Define as isotopy (isomorphism) class the subset of Ω_n formed by a particular Latin square and all its isotopic (isomorphic) ones. The following theorems concern the classification of members of Ω_n.

Theorem 1.4 *Each isotopy class of Ω_n splits into disjoint isomorphism classes.*

Definition 1.8 *A set of Latin squares, which comprises all the members of an isotopy class together with their conjugates is called a* main *class of Latin squares.*

Theorem 1.5 *Ω_n splits into disjoint main classes and each main class is a union of isotopy classes.*

An equivalent statement is the following.

Corollary 1.6 *Each main class of Ω_n, and therefore Ω_n itself, splits into disjoint isomorphism classes.*

1.2.2 Mutually Orthogonal Latin Squares

Considering MOLS, a natural question is which is the largest cardinality of a set of MOLS of order n, usually denoted as $N(n)$.

Theorem 1.7 (**[60, Theorem 2.1]**) *There can be at most $n-1$ MOLS of order n.*

A set of $n-1$ MOLS of order n is also called a *complete* set of MOLS. This bound is attainable for prime powers, i.e. numbers $n = p^m$, where p is prime and $m \in \mathbb{Z}_+$. A conjecture states that this bound it attainable if and only if n is a prime power [60, p.38]. For nonprime powers, the problem is relaxed to that of finding recursive constructions of sets of MOLS of order n from sets of MOLS of lower orders. The most representative method applies the concept of *Kronecker* product [60, Section 2.3] of two Latin squares of different orders. The result is the following theorem.

Theorem 1.8 *If there is a pair of OLS of order m and a pair of OLS of order n, there exists a pair of OLS of order mn.*

Using this theorem, it is easy to show the existence of OLS for all n, except for the ones that are an odd multiple of 2, i.e. for all cases not included in Euler's conjecture. The results of [20] prove that Euler's conjecture is actually false for all n that are odd multiples of 2, except for $n = 2, 6$. Hence, we may state the following.

Theorem 1.9 (**[60, Theorem 2.9]**) *There exists a pair of OLS for all orders $n \in \mathbb{Z}_+ \backslash \{1, 2, 6\}$.*

It follows that $N(n) \geq 2$ for all $n \in \mathbb{Z}_+ \backslash \{1, 2, 6\}$. Extending the method of *Kronecker* product, two more general statements become possible.

Theorem 1.10 *Let $q_1 \times \dots \times q_r$ be the factorisation of n into distinct prime powers, with $q_1 <, \dots, < q_r$. Then $N(n) \geq q_1 - 1$.*

Theorem 1.11 *For $m, n > 1$, $N(mn) \geq min\{N(m), N(n)\}$.*

Numerous recursive constructions of sets of MOLS can be found in [35, Chapter 5].

Research has also been conducted about criteria determining whether a set of k MOLS is extendible to a set of $k+1$ MOLS. A generalisation of Theorem 1.2 is the following.

Theorem 1.12 *k MOLS of order n are extendible to $k+1$ MOLS of order n, if they have n disjoint transversals in common.*

In [35, Sections 2.4-2.6], the reader may find more elegant considerations of the subject. We only state a simple, yet important, result.

Theorem 1.13 *A set of $n-3$ MOLS of order n is always extendible to a complete set of MOLS.*

Given that the existence of OLS has been resolved for all n, the next topic to be addressed is the existence of triples of MOLS. Quoting from [60, Section 16.5], it is known that there are at least 3 MOLS for all n, such that $4 \leq n \leq 10^4$, with the exception of $n = 6$ and, possibly, $n = 10$. The existence of 3 MOLS of order 10 is the most persistent unsolved instance, exactly because 10 is the smallest order for which there exists no complete set of MOLS ([58]). It is also the smallest order violating Euler's conjecture.

1.3 Related areas

Research on Latin squares is intensive not only because these structures give rise to interesting puzzles, but mainly because of their relevance to core areas of discrete mathematics and finite algebra. This section presents equivalences between complete sets of MOLS and certain combinatorial objects.

To begin our discussion, let us present the definition of an affine plane.

Definition 1.9 *An* affine plane *is a geometric system of points and lines, satisfying the following axioms*

A1 *There is a unique line joining any two points.*

A2 *For a certain point P and a line l not containing P, there is a unique line containing P and not intersecting l.*

A3 *There are four points, no three of which are on the same line.*

Two important corollaries are the following.

Corollary 1.14 *In any finite affine plane, there exists a positive integer n, such that every line contains exactly n points and every point is on exactly $n+1$ lines. Number n is said to be the* order *of the affine plane*

Corollary 1.15 *A finite affine plane of order n contains n^2 points.*

The fundamental link between finite affine planes and sets of MOLS is revealed by the next theorem.

Theorem 1.16 *A complete set of MOLS of order n exists if and only if a finite affine plane of order n exists.*

Hence, it follows directly that an affine plane of order 6 does not exist. A related concept is that of a projective plane.

Definition 1.10 *A* projective plane *is a geometric system of points and lines, satisfying the following axioms*

P1 *There is a unique line joining any two points.*

P2 *Any two lines intersect at a unique point.*

P3 *There are four points, no three of which are on the same line.*

We omit the counterparts of Corollaries 1.14 and 1.15 for projective planes and state directly the main result.

Theorem 1.17 *A finite affine plane of order n exists if and only if a finite projective plane of order n exists.*

Corollary 1.18 *A complete set of MOLS of order n exists if and only if a finite projective plane of order n exists.*

Hence, a problem regarding a complete set of MOLS can be tackled by any of its equivalents (see [58]) and vice versa.

Another related area of combinatorics is (t, m, s)-nets. The study of these structures is motivated by the necessity to approximate the value of a definite integral in cases where the anti-derivative of the integrand cannot be determined or does not exist. Again, we deliberately omit intermediate definitions, which are not related to our purposes. An excellent introduction appears in [60, Chapter 15]. Consider only that parameters t, m, s are sufficient to define the properties of a net.

Theorem 1.19 *Let $s \geq 2, b \geq 2$ be integers. There exists a $(0, 2, s)$-net in base b if and only if there exist s MOLS of order b.*

A more generic object in combinatorics is that of a design.

Definition 1.11 *Assume a set $\mathbb{P}$ of u points and a collection $\mathbb{B}$ of subsets, called blocks. The system of points and blocks is called a* design.

If we focus only on blocks of cardinality k, we restrict ourselves to $\binom{u}{k}$ blocks. If only $b < \binom{u}{k}$ blocks are included, the design is called *incomplete*. To obtain *balance*, we require that *(a)* each pair of points occurs together in a fixed number λ of blocks and *(b)* each point occurs in exactly r blocks, where $r \leq b$. Based on these involved preliminaries, we may define the structure of a *balanced incomplete block design* or a (u, b, r, k, λ)-design. Blocks and points are naturally mapped to lines and points of a geometric structure.

Lemma 1.20 *A finite affine plane of order n gives a (u, b, r, k, λ)-design, where $u = n^2$, $b = (n+1)n$, $k = n$, $r = n + 1$ and $\lambda = 1$.*

1.4 Applications of Orthogonal Latin Squares

Apart from their theoretical properties, MOLS possess diverse applications. Single Latin squares are applicable to even more areas. However, we will mention only the applications they share with MOLS.

The most well-known application of MOLS is multivariate statistical design. These designs are devised in order to determine whether significant differences exist between samples representing various levels of certain random variables. We present the example given in [60, Section 12.3], where n varieties of wheat need to be tested with n fertilisers and n insecticides. The inherent assumption is that all three variables are independent and have exactly the same number of levels, i.e. n. In order to determine the extent to which the wheat crops are affected by each fertiliser and insecticide, all n^3 triples of the form {wheat, fertiliser, insecticide} should be examined. To reduce the cost of the experiment, we may instead test each ordered pair of any two variables exactly once. This would imply n^2 repetitions of the experiment, instead of n^3. For $n = 4$, consider the first Latin square of Table 1.1 and let its rows correspond to different fertilisers and its columns to different insecticides. If the elements of each cell correspond to a different wheat variety, it can be seen that each wheat variety is tested exactly once with each fertiliser and each insecticide. Designing such an experiment for t statistical variables requires the use of a set of $t - 2$ MOLS of order n. Because of Theorem 1.7, it must be that $t - 2 \leq n - 1$ or $t \leq n + 1$. For example, there can exist no experiment of 4 variables having less than 3 levels each.

A relevant application is the design of tournaments. A pair of OLS of order n can be used to schedule any tournament between two teams, of n players each, with the following requirements:

- every player from team A plays against each player of team B;
- there are no games played between any two players of the same team;
- every player plays in each of the n rounds;
- every player plays at each of the n locations exactly once during the tournament.

To illustrate this via the OLS of Table 1.1, suppose that rows and columns correspond to players of teams A and B, respectively. Values of the first square indicate the round, where each pair meets, while values of the second square indicate the location. It is easy to see that numerous scheduling problems could be encompassed in the structure of MOLS.

MOLS have important applications in coding theory. A code is defined as a set of n-tuples, or vectors with n coordinates, where each coordinate belongs to a set S of q elements. This defines a *q-ary code of length n.* A code can be used to send messages, be it via a communication channel or within a computer system. The n-tuples recognised as valid messages by the code are its *codewords.* The objective when designing a code is to allow for a large number of codewords, which differ from each other sufficiently for errors to be efficiently detected and corrected. The larger the *Hamming distance d* between any codewords (for definitions see [35, Chapter 9]) the easier to detect errors and the smallest the maximum number of codewords for certain values of n and q. The notion of an (n, M, d) q-ary code C implies a code of length n with M codewords and minimum distance between any codewords equal to d. Fixing the values of n, q, d, the problem is to find a code with the maximum number of codewords M. Parameter n could be the bit-length of a network, q the number of states allowed for each bit and d the desired security level. The following theorem establishes the significance of MOLS for the existence of optimal codes.

Theorem 1.21 *There exists a q-ary $(s, q^2, s-1)$ code if and only if there exist $s-2$ MOLS of order q.*

Additional optimisation problems in codes and cryptography, which are related to MOLS, are described in detail in [35, Chapter 9] and [60, Chapter 13]. In fact, the list of application of MOLS presented here is only a representative sample. Extensive references for miscellaneous applications can be found in [60, Chapter 16]. The critical aspect is that all applications ask for a single pair of OLS or a single set of $k > 2$ MOLS.

1.5 IP models for the OLS problem

This section introduces two Integer Programming models for the OLS problem. Let the two squares be L_1, L_2 and consider four n-sets I, J, K, L. Suppose that I is the row set and J the column set, while sets K and L are the sets of values for the squares L_1 and L_2, respectively. Without loss of generality, it is convenient to assume that $I = J = K = L = \{0, ..., n-1\}$. Let variable x_{ijk} be 1 if value k appears in cell (i, j) of square L_1. Variable y_{ijl} is defined similarly. Since L_1 must be Latin, each value k must appear exactly once in all n cells of column j, i.e. in all cells defined by different values of i. Hence, we obtain the equality constraint $\sum\{x_{ijk} : i \in I\} = 1$ for all $j \in J$ and $k \in K$. Repeating the same argument for the fact that each value k must appear exactly once in each row

i and each cell (i,j) must contain exactly one value k gives rise to two more constraint types. The result is constraints (1.1)-(1.3) of the model depicted below. Notice that the roles of sets I, J, K, L are purely conventional. Constraints (1.4)-(1.6) ensure that L_2 is also a Latin square. To enforce the orthogonality condition, we introduce variables z_{ijkl}. Constraints (1.7)-(1.9) guarantee that z_{ijkl} will be 1 if and only if both variables x_{ijk} and y_{ijl} are 1, i.e. if pair (k,l) appears in cell (i,j). Finally, constraints (1.10) model the fact that each ordered pair (k,l) must occur exactly once.

$$min \sum\{z_{ijkl} : i \in I, j \in J, k \in K, L \in L\}$$

$$s.t. \quad :$$

$$\sum\{x_{ijk} : i \in I\} = 1, \forall j \in J, k \in K \tag{1.1}$$

$$\sum\{x_{ijk} : j \in J\} = 1, \forall i \in I, k \in K \tag{1.2}$$

$$\sum\{x_{ijk} : k \in K\} = 1, \forall i \in I, j \in J \tag{1.3}$$

$$\sum\{y_{ijl} : i \in I\} = 1, \forall j \in J, l \in L \tag{1.4}$$

$$\sum\{y_{ijl} : j \in J\} = 1, \forall i \in I, l \in L \tag{1.5}$$

$$\sum\{y_{ijl} : l \in L\} = 1, \forall i \in I, j \in J \tag{1.6}$$

$$x_{ijk} \geq z_{ijkl}, \forall i \in I, j \in J, k \in K, l \in L \tag{1.7}$$

$$y_{ijl} \geq z_{ijkl}, \forall i \in I, j \in J, k \in K, l \in L \tag{1.8}$$

$$x_{ijk} + y_{ijl} \leq z_{ijkl} + 1, \forall i \in I, j \in J, k \in K, l \in L \tag{1.9}$$

$$\sum\{z_{ijkl} : i \in I, j \in J\} = 1, \forall k \in K, l \in L \tag{1.10}$$

$$x_{ijk}, y_{ijl} \in \{0,1\}, \forall i \in I, j \in J, k \in K, l \in L$$

$$0 \leq z_{ijkl} \leq 1, \forall i \in I, j \in J, k \in K, l \in L$$

This is a model on $2n^3$ binary variables and n^4 real variables, which includes $7n^2 + 3n^4$ constraints.

An alternative model can be derived by directly defining $0-1$ variables x_{ijkl}. A variable x_{ijkl} is 1 if pair (k,l) appears in cell (i,j) and 0 otherwise. Since each pair must occur exactly once among all cells (i,j), the constraint $\sum\{x_{ijkl} : i \in I, j \in J\} = 1$ is valid for all $k \in K$ and $l \in L$. By taking into account that the roles of sets is conventional, therefore interchangeable, we can define 5 more analogous constraints. The model illustrated below is attributed to D.Gale (in [29]).

$$\sum\{x_{ijkl} : i \in I, j \in J\} = 1, \forall k \in K, l \in L \tag{1.11}$$

$$\sum\{x_{ijkl} : i \in I, k \in K\} = 1, \forall j \in J, l \in L \tag{1.12}$$

$$\sum\{x_{ijkl} : i \in I, l \in L\} = 1, \forall j \in J, k \in K \tag{1.13}$$

$$\sum\{x_{ijkl} : j \in J, k \in K\} = 1, \forall i \in I, l \in L \tag{1.14}$$

$$\sum\{x_{ijkl} : j \in J, l \in L\} = 1, \forall i \in I, k \in K \tag{1.15}$$

$$\sum\{x_{ijkl} : k \in K, l \in L\} = 1, \forall i \in I, j \in J \tag{1.16}$$

$$x_{ijkl} \in \{0,1\} \forall i \in I, j \in J, k \in K, l \in L$$

The objective function is equal to the sum of all the variables. This implies that every feasible solution is also optimal. Minimising or maximising this objective function is not really important, since our research treats *OLS* as a feasibility problem. This second model involves n^4 binary variables and $6n^2$ constraints.

Comparing the two models, it is clear that the first model is larger in terms of both variables and constraints. It follows that the associated linear programming relaxation would take longer to solve. Moreover, constraints of a form similar to that of (1.7) are known to produce very weak relaxations, as argued in [68, I.1]. However, the first model has far fewer integer variables, i.e. n^3 against n^4 of the second model. Thinking in terms of a *Branch & Bound* algorithm, it is reasonable to suggest that the second model is more impractical. However, this is not the case. Roughly speaking, the reason is simply that every integer feasible solution to the first model would require $2n^2$ integer variables to be set to 1, in contrast to the n^2 variables required to be 1 by the second model.

Another advantage of the second model is its simplicity and symmetry. This symmetry reflects more accurately the inherent symmetry of the OLS problem, in terms of the fact that all four entities are indistinguishable. Hence, the second model is the one used throughout the remainder of this thesis.

1.6 The convex hull of the OLS polytope for $n = 2$

The *convex hull* of a set of vectors $x_1, ..., x_m$ is defined as the set of all real points x, for which there exists non-negative scalars $\lambda_1, ...\lambda_m$, such that $x = \sum_{i=1}^{m} \lambda_i x_i$ and $\sum_{i=1}^{m} \lambda_i = 1$. Consider the second IP model of the previous paragraph and recall that the integer vectors satisfying constraints (1.11)-(1.16) have a $1-1$ correspondence to all pairs of OLS of order n. If we replace '=' by '$\leq$' in all constraints, we obtain a model, whose solutions are all incomplete OLS of order n. Notice that the zero vector, i.e. an "empty" pair of OLS, is also a solution to this model. This section illustrates all inequalities defining the convex hull of the vectors corresponding to incomplete OLS structures of $n = 2$. A pair of incomplete OLS of order 2 is illustrated at Table 1.4.

Table 1.4: A pair of incomplete OLS of order 2

0	1

1	0

To derive this convex hull we use the PORTA software ([24]), which accepts as input a set of integer vectors. The output is a minimal representation of the convex hull of the input vectors in terms of inequalities, which are *facet-defining.* The inequalities illustrated for this simple example anticipate the results of Chapter 2 and could also serve the purposes of an introduction to that chapter. Any terminology used to characterise each set of inequalities, e.g. cliques of type II, will be formally defined in Chapter 2.

Let A denote the constraint matrix of equalities (1.11)-(1.16). The convex hull of incomplete pairs of OLS is the polytope $\tilde{P}_I = conv\{x \in \{0,1\}^{n^4} : Ax \leq e\}$. Let $R = (K \times L) \cup \cdots \cup (I \times K)$ and $C = I \times J \times K \times L$ be the row and column sets of A, respectively. The facet-defining inequalities of $\tilde{P}_I$ for $n = 2$ are categorised as follows.

I) Non-negativities

These are defined for all $c \in C$. For $c = (i_0, j_0, k_0, l_0)$, the form of such an inequality is:

$$-x_{i_0 j_0 k_0 l_0} \leq 0$$

There exist n^4 such inequalities, or 16 for $n = 2$.

II) Cliques of type I

These are defined for any $r \in R$. For $r = (i_0, j_0)$, the form of such an inequality for $n = 2$ is:

$$x_{i_0 j_0 k_0 l_0} + x_{i_0 j_0 k_0 l_1} + x_{i_0 j_0 k_1 l_0} + x_{i_0 j_0 k_1 l_1} \leq 1$$

There are $6n^2$ such inequalities, or 24 for $n = 2$.

III) Cliques of type II

These are defined for all $c \in C$. For $c = (i_0, j_0, k_0, l_0)$, the form of such an inequality for $n = 2$ is:

$$x_{i_0 j_0 k_0 l_0} + x_{i_1 j_0 k_0 l_0} + x_{i_0 j_1 k_0 l_0} + x_{i_0 j_0 k_1 l_0} + x_{i_0 j_0 k_0 l_1} \leq 1$$

Hence, the left-hand side includes variable x_c and all variables having exactly three indices in common with x_c. There are n^4 such inequalities, or 16 for $n = 2$.

IV) Cliques of type III

These are defined for all $c, d \in C$ such that $|c \cap d| = 1$. Assuming $c = (i_0, j_0, k_0, l_0)$, $d = (i_0, j_1, k_1, l_1)$, such an inequality has the form:

$$x_{i_0 j_0 k_0 l_0} + x_{i_0 j_0 k_1 l_1} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_1 k_1 l_0} \leq 1$$

Notice that the left-hand side includes variable x_c along with all variables sharing two indices with x_c and three indices with x_d. There are $n^4(n-1)^3$ such inequalities, or 16 for $n = 2$.

V) Composite cliques

These are defined for all $c, d \in C$ such that $|c \cap d| = 0$. For $c = (i_0, j_0, k_0, l_0)$, $d = (i_0, j_1, k_1, l_1)$, the form of such an inequality is:

$$2 \cdot x_{i_0 j_0 k_0 l_0} + x_{i_1 j_0 k_0 l_0} + x_{i_0 j_1 k_0 l_0} + x_{i_0 j_0 k_1 l_0} + x_{i_0 j_0 k_0 l_1} +$$
$$x_{i_0 j_0 k_1 l_1} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_1 k_1 l_0} + x_{i_1 j_0 k_0 l_1} + x_{i_1 j_0 k_1 l_0} + x_{i_1 j_1 k_0 l_0} \leq 2$$

The left-hand side includes x_c and all variables having two or three indices in common with x_c. There are $n^4(n-1)^4$ such inequalities, or 16 for $n = 2$.

VI) Lifted antiwebs

These are defined for all $c, d \in C$ such that $|c \cap d| = 0$. For $c = (i_0, j_0, k_0, l_0)$, $d = (i_0, j_1, k_1, l_1)$, the form of such an inequality is:

$$\begin{aligned}
&x_{i_0 j_0 k_0 l_0} + x_{i_0 j_0 k_0 l_1} + x_{i_0 j_0 k_1 l_0} + x_{i_0 j_0 k_1 l_1} + \\
&x_{i_0 j_1 k_0 l_0} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_1 k_1 l_0} + x_{i_0 j_1 k_1 l_1} + \\
&x_{i_1 j_0 k_0 l_0} + x_{i_1 j_0 k_0 l_1} + x_{i_1 j_0 k_1 l_0} + x_{i_1 j_0 k_1 l_1} + \\
&x_{i_1 j_1 k_0 l_0} + x_{i_1 j_1 k_0 l_1} + x_{i_1 j_1 k_1 l_0} + x_{i_1 j_1 k_1 l_1} \leq 2
\end{aligned}$$

Observe that the left-hand side includes variables x_c and x_d.along with all variables having all their indices belonging to c or d. There are $\frac{1}{16}n^4(n-1)^4$ such inequalities, or a single one for $n = 2$.

The last inequality states that there can exist no pair of OLS for $n = 2$. The total number of inequalities defining $\tilde{P}_I$ is 89. As mentioned earlier, all these inequalities, along with further inequalities that appear only for larger values of n, are analysed in the next chapter.

Chapter 2

The intersection graph of the OLS problem

This chapter presents a framework for characterising the OLS problem from an *Integer Programming* (IP) perspective. A first step towards this direction is the formulation of an IP model for the problem, i.e. a linear model on integer variables. Such a model was presented in Section 1.5. A relaxation of this model is derived by requiring the variables to be simply non-negative. This is the problem's *linear programming* (LP) relaxation. The convex hull of the feasible solutions of the original model defines a polytope P_I, within the polytope P_L defined by the problem's LP relaxation. Although the feasible points are located within P_I, we initially know only the set of linear inequalities defining P_L.

In order to efficiently characterise and solve the OLS problem using IP, at least a partial knowledge of polytope P_I is required. This knowledge is usually provided in the form of inequalities, which are valid for polytope P_I but not for the polytope P_L. In other words, these inequalities are derived by enforcing the requirement that certain variables must be integer. Among these inequalities, it is reasonable to identify the ones, which are not dominated by any other valid inequality. These valid inequalities are called *maximal.* Any maximal valid inequality defines a non-empty face of P_I and the set of maximal valid inequalities contains all of the facet-defining inequalities for P_I ([68, p.207]).

Amongst a number of methods for identifying families of valid inequalities, one with a considerable record, especially in problems involving a $0-1$ constraint matrix, is that of studying a graph associated with the IP model. This is the *intersection graph,* introduced in Section 2.2 (see also [26, 69]). Subgraphs of this graph exhibiting a particular structure give rise to strong valid inequalities for the *set-packing* and *set-partitioning* polytopes (see [8, 18] for a review). In this chapter, a number of such subgraphs, along with the induced inequalities, are identified and analysed for the intersection graph of our IP model. In particular, cliques, antiwebs, wheels, composite cliques and odd holes are presented in sections 2.3-2.7, respectively. Finally, Section 2.8 examines the existence of odd antiholes and proves that such structures do not exist for this particular problem.

2.1 The OLS polytope

Recall the model introduced in Section 1.5:

$$\sum\{x_{ijkl} : i \in I, j \in J\} = 1, \forall k \in K, l \in L \tag{2.1}$$

$$\sum\{x_{ijkl} : i \in I, k \in K\} = 1, \forall j \in J, l \in L \tag{2.2}$$

$$\sum\{x_{ijkl} : i \in I, l \in L\} = 1, \forall j \in J, k \in K \tag{2.3}$$

$$\sum\{x_{ijkl} : j \in J, k \in K\} = 1, \forall i \in I, l \in L \tag{2.4}$$

$$\sum\{x_{ijkl} : j \in J, l \in L\} = 1, \forall i \in I, k \in K \tag{2.5}$$

$$\sum\{x_{ijkl} : k \in K, l \in L\} = 1, \forall i \in I, j \in J \tag{2.6}$$

$$x_{ijkl} \in \{0,1\} \forall i \in I, j \in J, k \in K, l \in L \tag{2.7}$$

where I, J, K, L are disjoint sets with $\mid I \mid = \mid J \mid = \mid K \mid = \mid L \mid = n$.

Given real weights c_{ijkl} for every $(i,j,k,l) \in I \times J \times K \times L$, the problem of minimizing the function $\sum\{c_{ijkl}x_{ijkl} : i \in I, j \in J, k \in K, l \in L\}$ over the polytope described by constraints (2.1),..., (2.7) is the *planar 4-index assignment* problem or $4PAP_n$ ([2]), which is essentially the optimisation version of the *OLS* problem. To formally define the $4PAP_n$ consider the n^3 triples formed by selecting a single element from each of the sets I, J, K. Assume that sets I, J, K, L index the rows, columns and values of squares L_1 and L_2, respectively. Under this convention, notice that a transversal of square L_1 corresponds to n disjoint triples (Definition 1.6). For example, a transversal of square L_1 in Table 1.1 is defined by the triples $\{(0,0,0), (1,2,3), (2,3,1), (3,1,2)\}$. Hence, the $4PAP_n$ asks for a minimum weight collection of n^2 triples, which form n disjoint sets of n disjoint triples. Set L can be regarded as indexing the n disjoint sets of triples. It is easy to see that, according to Theorem 1.2, the $4PAP_n$ asks for a minimum weight pair of *OLS*.

Let A denote the coefficient matrix of constraints (2.1),..., (2.6). We define the polytope $P_L = \{x \in \mathbb{R}^{n^4} : Ax = e, x \geq 0\}$ where $e = \{1,1,...,1\}^T \in \mathbb{R}^{6n^2}$. The convex hull of integer vectors of P_L is defined as $P_I = \text{conv}\{x \in \{0,1\}^{n^4} : Ax = e\}$. This is the *OLS* polytope since every integer point $x \in P_I$ is a pair of *OLS*. P_L is also called the *linear relaxation* of P_I. Clearly, $P_I \subset P_L$ and the extreme points of P_I are among the extreme points of P_L. We will sometimes refer to P_I as P_I^n in order to include the concept of order in the notation. Thus, $P_I^6 = \emptyset$ is another way of stating Euler's conjecture for $n = 6$.

Substituting $(=)$ by $(\leq)$ in constraints (2.1),..., (2.6) yields the polytope $\tilde{P}_I = \text{conv}\{x \in \{0,1\}^{n^4} : Ax \leq e\}$. The convex hull of $\tilde{P}_I$, for $n = 2$, was presented in Section 1.6. Polytopes $P_I, \tilde{P}_I$ are related since $P_I \subset \tilde{P}_I$. Hence, facets of $\tilde{P}_I$ are also valid inequalities, but not necessarily facets, for P_I. If D denotes a $0-1$ matrix, it can be seen that P_I is a special case of the set-partitioning polytope $P_{SPP} = \{x \in \{0,1\}^q : Dx = e\}$, while $\tilde{P}_I$ is a special case of the set-packing polytope $P_{SP} = \{x \in \{0,1\}^q : Dx \leq e\}$ (see [8, 69] for details).

There are two problems, each involving three disjoint n-sets, that are highly related to the *OLS* problem: the *planar three-index assignment problem* $(3PAP_n)$ and the *axial three-index assignment problem* $(3AAP_n)$. Both are defined with respect to three disjoint $n-$ sets, namely I, J, K, and a weight coefficient c_{ijk} for each triplet $(i,j,k) \in I \times J \times K$. The $3AAP_n$ is the problem of finding n disjoint triplets (i.e. a transversal) of minimum weight and constitutes an extension of the (two-index) assignment problem or *weighted bipartite matching* problem ([68]). The IP formulation of

$3AAP_n$ is:

$$\begin{aligned} & min \sum\{c_{ijk} \cdot x_{ijk} : i \in I, j \in J, k \in K\} & \\ s.t. \quad & \sum\{x_{ijk} : i \in I, j \in J\} = 1, \forall k \in K & (2.8) \\ & \sum\{x_{ijk} : i \in I, k \in K\} = 1, \forall j \in J & (2.9) \\ & \sum\{x_{ijk} : j \in J, k \in K\} = 1, \forall i \in I & (2.10) \\ & x_{ijk} \in \{0,1\} \forall i \in I, j \in J, k \in K & (2.11) \end{aligned}$$

The $3PAP_n$ is the problem of finding a minimum weight collection of n^2 disjoint pairs, forming n sets of n disjoint pairs each. It is equivalent to the problem of finding a minimum weight Latin squares of order n (see [37]). Its IP formulation is the following.

$$\begin{aligned} & min \sum\{c_{ijk} \cdot x_{ijk} : i \in I, j \in J, k \in K\} & \\ s.t. \quad & \sum\{x_{ijk} : i \in I\} = 1, \forall j \in J, k \in K & (2.12) \\ & \sum\{x_{ijk} : j \in J\} = 1, \forall i \in I, k \in K & (2.13) \\ & \sum\{x_{ijk} : k \in K\} = 1, \forall i \in I, j \in J & (2.14) \\ & x_{ijk} \in \{0,1\} \forall i \in I, j \in J, k \in K & (2.15) \end{aligned}$$

Polyhedral analysis of $3AAP_n$ and $3PAP_n$ can be found in [9, 10, 76, 38] and [37], respectively. We extend this analysis in the general setting of multi-index assignment problems in Chapter 7.

2.2 The intersection graph

Let R and C denote the index sets of rows and columns, respectively, of the $0-1$ A matrix. We refer to a column of the A matrix as a^c for $c \in C$. The *(column) intersection* graph $G_A(V, E)$ of a 0-1 A matrix has a node c for every column a^c of A and an edge (c_s, c_t) if and only if $a^{c_s} \cdot a^{c_t} \geq 1$, i.e. both columns c_s and c_t of A have a +1 entry in at least one common row ([69]).

Let $G_A(C, E_C)$ denote the intersection graph of *OLS*, where $C = I \times J \times K \times L$. It is convenient to label the n^4 columns of the *OLS* A matrix not from 1 to n^4, but with four indices i, j, k and l ranging from 1 to n. In this sense, node c_s of G_A represents the index set of column s.

Definition 2.1 *The intersection graph of* OLS $G_A = (C, E_C)$ *has a node c, for every* $c \in C$, *and an edge* (c_s, c_t) *for every pair of nodes* $c_s, c_t \in C$ *such that* $\mid c_s \cap c_t \mid= 2$ *or* 3.

Note that an edge $(c_s, c_t) \in E_C$ corresponds to columns a^{c_s}, a^{c_t} with $a^{c_s} \cdot a^{c_t} = 1$ or 3. The row set of the *OLS* A matrix is defined as $R = (K \times L) \cup (I \times L) \cup (J \times L) \cup (I \times J) \cup (J \times K) \cup (I \times K)$. Since $\mid I \mid=\mid J \mid=\mid K \mid=\mid L \mid= n$, $\mid C \mid= n^4$ and $\mid R \mid= 6n^2$. Notice also that two nodes are connected if and only if they have two or three indices in common. The edge connecting any two nodes can be characterised by the indices these two nodes have in common. Hence, an edge (c_s, c_t) can be based on either a double or a triple ground set, therefore regarded as a double or a triple link, respectively. We write $(c_s, c_t) \in M_1 \times M_2$ or $(c_s, c_t) \in M_1 \times M_2 \times M_3$, where M_1, M_2, M_3 can be any of the sets I, J, K, L, to denote that $\mid c_s \cap c_t \mid$ 2 or 3, respectively. When illustrating an edge (c_s, c_t), the indices, which are said to "appear" at the edge, are exactly the members of the subset $c_s \cap c_t$.

Proposition 2.1 *The graph $G_A(C, E_C)$ is regular of degree $2(3n-1)(n-1)$.*

Proof. Consider any $c \in C$. There are $(n-1)^4$ elements of C, which have no index in common with c. For each of the four indices of c there are $(n-1)^3$ elements of C, which share the same value for this index but have different values for the other three. Therefore, there are $4(n-1)^3$ elements of C which have exactly one index in common with c. By Definition 2.1, c is connected only to nodes that have two or three indices in common with it, so it is connected to all but $(n-1)^4 + 4(n-1)^3$ nodes. Therefore the degree of each $c \in C$ is $n^4 - 1 - ((n-1)^4 + 4(n-1)^3) = 6n^2 - 8n + 2 = 2(3n-1)(n-1)$. ■

Corollary 2.2 $\mid E_C \mid = n^4(3n-1)(n-1)$.

Proof. Since the number of edges of a graph equals the sum of the degrees of its nodes divided by 2, it follows that $\mid E_C \mid = 0.5 \times n^4 \times 2(3n-1)(n-1) = n^4(3n-1)(n-1)$. ■

2.3 The cliques of G_A

A *maximal complete* subgraph of a graph $G(V, E)$ is called a *clique* ([9, 69]). Let $Q \subseteq V$ denote the node set of a clique. The *cardinality* of a clique is the cardinality of its node set Q, denoted $\mid Q \mid$.

Let a_r^c denote the entry of the A matrix at row r and column c. Then we define the set $R(r) = \{c \in C : a_r^c = 1\}$. So $R(r)$ denotes the set of columns with a non-zero entry in row r.

Proposition 2.3 *For each $r \in R$, the node set $R(r)$ induces a clique of cardinality n^2 in $G_A(C, E_C)$. There are $6n^2$ cliques of this type.*

Proof. The subgraph induced by the node set $R(r)$ is complete since all its elements have two indices in common. To prove that it is also maximal assume w.l.o.g. that $r = (i_1, j_1) \in I \times J$ and consider $c_0 = (i_0, j_0, k_0, l_0) \in C \setminus R(r)$ where $i_0 \neq i_1$ and $j_0 \neq j_1$. Since $R(r)$ contains all n^2 elements of C, whose first two indices are i_1 and j_1, it contains an element $c_1 = (i_1, j_1, k_1, l_1)$ with $\mid c_0 \cap c_1 \mid = 0$. Next consider $c_0 = (i_1, j_0, k_0, l_0) \in C \setminus R(r)$. But then there exists $c_1 \in C$ (for example $c_1 = (i_1, j_1, k_1, l_1)$) so that $\mid c_0 \cap c_1 \mid = 1$. The same happens if $c_0 = (i_0, j_1, k_0, l_0)$. Therefore there is no c_0 such that the subgraph induced by $R(r) \cup \{c_0\}$ is complete. Consequently, the subgraph having $R(r)$ as its node set is maximal. There are as many cliques of this type as the number of rows of the A matrix, i.e. $6n^2$. ■

Proposition 2.4 *For each $c \in C$ the set $Q(c) = \{c\} \cup \{s \in C :\mid c \cap s \mid = 3\}$ induces a clique of cardinality $4n-3$ in $G_A(C, E_C)$. There are n^4 cliques of this type.*

Proof. W.l.o.g. consider $c = c_0 = (i_0, j_0, k_0, l_0) \in C$. Let $c_1, c_2 \in Q(c_0)$ with $c_1 \neq c_2 \neq c_0 \neq c_1$. Since c_1, c_2 have three indices in common with c_0, at least two of their indices coincide. Therefore, $(c_1, c_2) \in E_C$ for any $c_1, c_2 \in Q(c_0)$, i.e. $Q(c_0)$ is complete. To show that $Q(c_0)$ is also maximal consider $c_3 = (i_3, j_3, k_3, l_3) \in C \setminus Q(c_0)$. Then c_3 has exactly two indices in common with c_0, by definition. If $\mid c_0 \cap c_3 \mid = 2$, w.l.o.g. consider $i_0 = i_3$, $j_0 = j_3$ and $k_0 \neq k_3$, $l_0 \neq l_3$. By definition, $Q(c_0)$ contains two elements, namely $c_s = (i_s, j_0, k_0, l_0)$ and $c_t = (i_0, j_t, k_0, l_0)$ such that $i_0 \neq i_s$ and $j_0 \neq j_t$. But then $\mid c_3 \cap c_s \mid = \mid c_3 \cap c_t \mid = 1$, i.e. the subgraph $Q(c_0) \cup \{c_3\}$ is not complete. Hence, $Q(c_0)$ is also maximal.

The set $Q(c_0)$ includes node $c_0 = (i_0, j_0, k_0, l_0)$ and all nodes with exactly one index different from c_0, hence $\mid Q(c_0) \mid = 4(n-1) + 1 = 4n - 3$. There are n^4 elements belonging to the set C and each one can play the role of c_0. Therefore, there are n^4 distinct cliques of this type. ■

Proposition 2.5 *Let $c, s \in C$ such that $\mid c \cap s \mid = 1$. Then the set $Q(c,s) = \{c\} \cup \{t \in C : \mid c \cap t \mid = 2, \mid s \cap t \mid = 3\}$ induces a 4-clique in $G_A(C, E_C)$.*

Proof. W.l.o.g. let $c = c_0 = (i_0, j_0, k_0, l_0)$ and $s = (i_0, j_1, k_1, l_1)$. We can uniquely define three elements $t_1 = (i_0, j_0, k_1, l_1)$, $t_2 = (i_0, j_1, k_0, l_1)$, $t_3 = (i_0, j_1, k_1, l_0)$, satisfying $\mid c \cap t_i \mid = 2$ and $\mid s \cap t_i \mid = 3$ for $i = 1, 2, 3$. It is obvious that the node set $\{c, t_1, t_2, t_3\}$ induces a complete subgraph of $G_A(C, E_C)$. To show that this subgraph is also maximal, consider $c_2 = \{i_2, j_2, k_2, l_2\} \in C \setminus Q(c,s)$. If $i_2 \neq i_0$ then for an edge (c, c_2) to exist in $G_A(C, E_C)$ we must have $\mid c \cap c_2 \mid \geq 2$, which implies that $\mid c_2 \cap t_i \mid = 1$ for $i = 1, 2, 3$. Therefore, $Q(c,s)$ cannot be extended to include c_2, since the resulting graph is not complete. If $i_2 = i_0$ either c_2 has another element common with c and the remaining two with s, in which case it coincides with one of the t_i's, or it has one more element in common with c and with s. In the latter case, w.l.o.g. let $j_2 = j_0$ and $k_2 = k_1$. Then we have $\mid c_2 \cap t_2 \mid = 1$. Hence, $Q(c,s)$ cannot again be extended. The results follows. ■

Concerning the cardinality of the set of cliques of this type, every ordered pair (c,s) such that $\mid c \cap s \mid = 1$ can be used to create a clique of this type. Considering that $\mid C \mid = n^4$ and that for each $c \in C$ there are $4(n-1)^3$ possible s such that $\mid c \cap s \mid = 1$, the number of such ordered pairs is $4n^4(n-1)^3$. Note, however, that the 4-clique $Q(c,s) = (c, t_1, t_2, t_3)$ is also generated as $Q(c_i, s_i)$ for $i = 1, 2, 3$ where

$c_1 = t_1 = (i_0, j_0, k_1, l_1)$ and $s_1 = (i_0, j_1, k_0, l_0)$,
$c_2 = t_2 = (i_0, j_1, k_0, l_1)$ and $s_2 = (i_0, j_0, k_1, l_0)$,
$c_3 = t_3 = (i_0, j_1, k_1, l_0)$ and $s_3 = (i_0, j_0, k_0, l_1)$.

Proposition 2.6 $Q(c,s) = Q(c_i, s_i)$, $i = 1, 2, 3$

It is also obvious that the 4-clique $Q(c,s) = (c, t_1, t_2, t_3)$ cannot arise from any other choice of c and s.

Corollary 2.7 *The number of distinct 4-cliques is $n^4(n-1)^3$.*

Proof. Each 4-clique arises from four different ordered pairs of C and there exist $4n^4(n-1)^3$ such pairs. ■

Cliques described in Propositions 2.3, 2.4 and 2.5 will be called cliques of type I, II and III respectively. Each clique Q of the intersection graph $G_A(V, E)$ defines an inequality of the form $\sum\{x_q : q \in Q\} \leq 1$. In particular, cliques of type II, for $c = (i_0, j_0, k_0, l_0)$, define an inequality of the form:

$$x_{i_0 j_0 k_0 l_0} + \sum_{i \neq i_0} x_{i j_0 k_0 l_0} + \sum_{j \neq j_0} x_{i_0 j k_0 l_0} + \sum_{k \neq k_0} x_{i_0 j_0 k l_0} + \sum_{l \neq l_0} x_{i_0 j_0 k_0 l} \leq 1$$

while cliques of type III, for $c = (i_0, j_0, k_0, l_0)$ and $s = (i_0, j_1, k_1, l_1)$ define the inequality:

$$x_{i_0 j_0 k_0 l_0} + x_{i_0 j_1 k_1 l_0} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_0 k_1 l_1} \leq 1$$

Inequalities arising from cliques of type I, taken as equalities, define the original formulation. Examples of these inequalities, for $n = 2$, were given in Section 1.6

Theorem 2.8 *The cliques of type I, II and III are the only cliques in $G_A(C, E_C)$.*

Proof. Let Q be the node set of a clique in $G_A(C, E_C)$. Let $c = (i_0, j_0, k_0, l_0) \in Q$. Every other $q \in Q$ must have at least two indices in common with c. If $\mid c \cap q \mid = 3$ for all $q \in Q$, Q is a node set of a clique of type II. Otherwise, there must exist $q_s \in Q$, such that $\mid c \cap q_s \mid = 2$. W.l.o.g. let $q_s = (i_0, j_0, k_1, l_1)$. If every other element of Q has the same values i_0, j_0 for the first two indices, Q is a node set of a clique of type I. If not, there must exist $q_t = (i_t, j_t, k_t, l_t) \in Q$, which satisfies all of the following conditions:

(i) Either $i_t = i_0$ or $j_t = j_0$. If both $i_t \neq i_0$ and $j_t \neq j_0$ then it must be $k_t = k_0$ and $l_t = l_0$ in order for q_t to be connected to c. But then $\mid q_s \cap q_t \mid = 0$, i.e. Q does not induce a clique.

(ii) Either $(k_t, l_t) = (k_0, l_1)$ or $(k_t, l_t) = (k_1, l_0)$. If both $k_t \neq k_0$ and $l_t \neq l_0$ then, together with condition (i), we have $\mid c \cap q_t \mid = 1$; if $k_t = k_0$ and $l_t = l_0$ then $\mid q_s \cap q_t \mid = 1$. In both cases, Q does not induce a clique.

W.l.o.g. assume that $q_t = (i_0, j_1, k_0, l_1)$. If $(i_0, j_1, k_1, l_0) \in Q$ then $Q \equiv Q(c, (i_0, j_1, k_1, l_1))$, i.e. Q is the node set of a clique of type III. If $(i_0, j_1, k_1, l_0) \notin Q$, there must exist $q_r \in Q$, such that $\mid q_r \cap (i_0, j_1, k_1, l_0) \mid \leq 1$, $\mid q_r \cap c \mid \geq 2$, $\mid q_r \cap q_s \mid \geq 2$ and $\mid q_r \cap q_t \mid \geq 2$. Every such q_r must have at least three indices in common with (i_0, j_0, k_0, l_1), in which case $Q \equiv Q(i_0, j_0, k_0, l_1)$, i.e. Q induces a clique of type II. ■

Corollary 2.9 *The total number of cliques in $G_A(C, E_C)$ is $n^4((n-1)^3 + 1) + 6n^2$.*

Proof. As shown above, there are $6n^2$, n^4, and $n^4(n-1)^3$ cliques of type I, II and III respectively. ■

It is easy to see that an inequality arising from a clique of G_A cannot be augmented by adding more variables with non-zero coefficients to its l.h.s. without raising its r.h.s. ([69]). It follows that every other inequality with a l.h.s. of 1 and positive coefficients at the l.h.s. is bound to be dominated by a clique inequality. Under this view, clique inequalities are the strongest inequalities with r.h.s. of 1.

2.4 A family of Antiwebs of G_A

An *antiweb* $AW(x, y)$ is a graph containing x nodes, each connected to all but y other nodes. A *web* is the complement of an antiweb. Web and antiweb structures were introduced in [82]. Figure 2.1 shows the generic structure of an $AW(8, 3)$ antiweb graph, where each node is connected to all but 3 nodes. Each antiweb $AW(x, y)$, $y < x$, gives rise to an inequality of the form $\sum \{x_i : i \in AW(x, y)\} \leq \left\lfloor \frac{x}{y} \right\rfloor$.

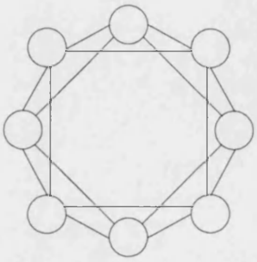


Figure 2.1: The antiweb AW(8,3)

Assume $c = (i_0, j_0, k_0, l_0)$, $d = (i_1, j_1, k_1, l_1)$ and consider the node set $A(c, d) = \{(i_0, j_0, k_0, l_0), (i_1, j_0, k_0, l_0), (i_1, j_1, k_0, l_0), (i_1, j_1, k_1, l_0), (i_1, j_1, k_1, l_1), (i_0, j_1, k_1, l_1), (i_0, j_0, k_1, l_1), (i_0, j_0, k_0, l_1)\}$.

This node set constitutes an AW(8,3), as depicted at Figure 2.2. Notice that every triple of consecutive nodes forms a complete subgraph. The inequality induced by $A(c,d)$ is:

$$x_{i_0j_0k_0l_0} + x_{i_0j_0k_0l_1} + x_{i_0j_0k_1l_1} + x_{i_0j_1k_1l_1} + x_{i_1j_1k_1l_1} + x_{i_0j_1k_1l_1} + x_{i_0j_0k_1l_1} + x_{i_0j_0k_0l_1} \leq 2 \quad (2.16)$$

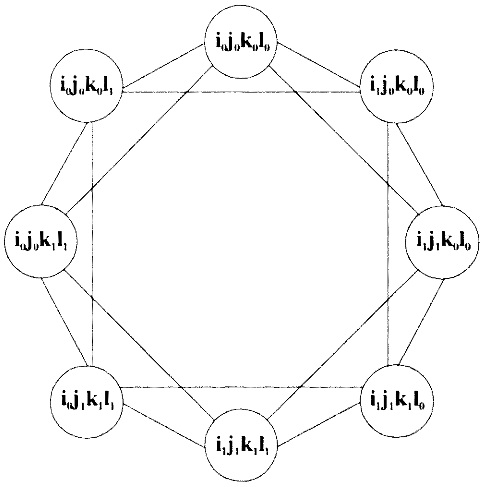


Figure 2.2: The antiweb A(c,d)

Inequality (2.16) can be further strengthened by applying the process of sequential lifting, as introduced at [69]. The aim is to include the maximum number of variables in the l.h.s. of (2.16), each one with the maximum possible coefficient, without raising the value of the r.h.s. The coefficient of an additional variable included in the lifting is trivially bounded by the value of the r.h.s.

This coefficient is also called the *lifting coefficient* for a variable. The order, according to which variables are considered during the lifting process, is called the *lifting sequence.* The lifting coefficient(s) are generally dependent on the lifting sequence (see [18, 69] for a discussion).

Assume variable x_s, $s \in C \backslash A(c,d)$ and let a_s be its coefficient in the lifted inequality:

$$a_s x_s + \sum_{t \in A(c,d)} x_t \leq 2$$

For $x_s = 0$, we get (2.16), whereas for $x_s = 1$ we get $\sum_{t \in A(c,d)} x_t \leq 2 - a_s$. It follows that $a_s = 2$, if s is connected to all 8 nodes of $A(c,d)$, since setting $x_s = 1$ would set to 0 all other variables. No node has this property. On the other hand, $a_s = 1$, if s is connected to at least 5 consecutive nodes of $A(c,d)$. In that case, setting $x_s = 1$ would set to 0 the variables corresponding to these 5 consecutive nodes and only one of the remaining variables could take value 1. The nodes having this property are $\overline{A(c,d)} = \{(i_0, j_1, k_0, l_0), (i_0, j_0, k_1, l_0), (i_0, j_1, k_0, l_1), (i_0, j_1, k_1, l_0), (i_1, j_0, k_1, l_1), (i_1, j_1, k_0, l_1), (i_1, j_0, k_1, l_0), (i_1, j_0, k_0, l_1)\}$. Note that this node set also forms an antiweb. Hence, at most 2 variables $x_s, s \in \overline{A(c,d)}$ can be set to 1. If exactly two variables $x_s, s \in \overline{A(c,d)}$ are set to 1, all other variables $x_s \in A(c,d) \cup \overline{A(c,d)}$ are set to 0 and vice versa. Therefore, the lifted antiweb inequality is:

$$\begin{aligned} & x_{i_0j_0k_0l_0} + x_{i_0j_0k_0l_1} + x_{i_0j_0k_1l_0} + x_{i_0j_0k_1l_1} + \ x_{i_0j_1k_0l_0} + x_{i_0j_1k_0l_1} + x_{i_0j_1k_1l_0} + x_{i_0j_1k_1l_1} & & (2.17) \\ & x_{i_1j_0k_0l_0} + x_{i_1j_0k_0l_1} + x_{i_1j_0k_1l_0} + x_{i_1j_0k_1l_1} + \ x_{i_1j_1k_0l_0} + x_{i_1j_1k_0l_1} + x_{i_1j_1k_1l_0} + x_{i_1j_1k_1l_1} & \leq & \ 2 \end{aligned}$$

More formally, for c, $d \in C$, $|c \cap d| = 0$, let $A(c,d) = \{f \in C : f \subseteq c \cup d\}$. Then the lifted antiweb inequality

$$X(A(c,d)) = \sum\{x_f : f \in \ A(c,d)\} \leq 2 \qquad (2.18)$$

is valid. Note that the form of the lifted inequality is independent of the lifting sequence.

This inequality states that no pair of OLS can exist for $n = 2$. Equivalently, it states that any pair of 2×2 subsquares in a pair of OLS of order n is bound to contain at least 3 distinct values.

Proposition 2.10 *The number of inequalities (2.18) is* $\frac{1}{16}n^4 \cdot (n-1)^4$.

Proof. Deriving an inequality (2.18) requires exactly two distinct indices from each of the sets I, J, K, L. The number of options for selecting two indices from each of the four sets are $\binom{n}{2}^4 = \left(\frac{n(n-1)}{2}\right)^4 = \frac{1}{16}n^4 \cdot (n-1)^4$. ■

2.5 The Wheels of G_A

Definition 2.2 *For $c \in C$ and an integer $p \geq 2$, consider the node set:*

$$\begin{aligned} H(c) \quad = \quad & \{c_i \in C, i = 0, ..., 2p : (c_s, c_t) \in E_c, \quad \text{for } s = t \pm 1(\text{mod}\, 2p+1\}, & (2.19) \\ & (c, c_s) \in E_c, \ \text{for all } s \in \{1, ..., 2p+1\}\} \end{aligned}$$

The node set $W(c) = \{c\} \cup H(c)$ is called a wheel.

Node c is the *hub* of the wheel, $H(c)$ is the *rim* and edges (c, c_s) are the *spokes*. Any other edge $(c_i, c_j), i \neq j, i \neq j \pm 1(\text{mod}\, 2p+1\}$ is called a chord. The rim is always assumed to be chordless. Figure 2.3 exhibits a wheel subgraph of G_A.

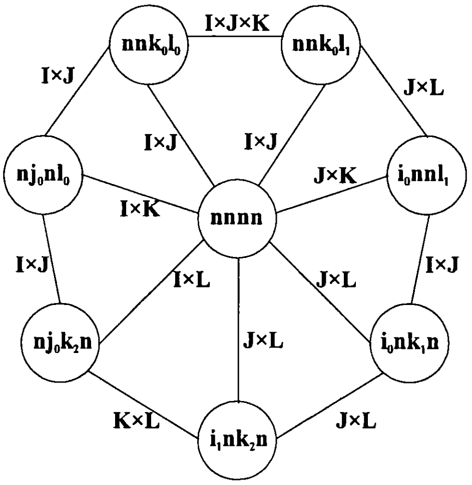


Figure 2.3: A wheel of size 7

For $i, j \in \{1, ..., 2p+1\}$, the spokes $(c, c_i), (c, c_j), i \neq j$, will be called *rim adjacent*, if they are incident to adjacent nodes of the rim, i.e. if $(c_i, c_j) \in E_C$. For example, spokes $((n,n,n,n), (n,n,k_0,l_1))$ and $((n,n,n,n), (i_0,n,n,l_1))$ in Figure 2.3 are rim adjacent. Two nodes of the rim, whose spokes are based on the same double ground set, will be called *conjugate*. An example of such a pair is nodes (n,n,k_0,l_0) and (n,n,k_0,l_1) in Figure 2.3. The rim edges can be either double or triple links, i.e. $|c_s, c_t| \geq 2$ for $c_s, c_t \in H(c)$ and $s \neq t$.We assume, however, that the spokes can only be double links, i.e. $|c \cap c_s| = 2$ for all $c_s \in H(c)$.Throughout the rest of this section, we also assume, w.l.o.g., that $c = (n,n,n,n)$ and $p \geq 2$.

Each wheel induces an inequality of the form:

$$p \cdot x_c + \sum_{d \in H(c)} x_d \leq p \tag{2.20}$$

The validity of this inequality is justified as follows: setting x_c=1 implies $x_d = 0$ for all $d \in H(c)$; if $x_c = 0$, at most p of the variables x_d, $d \in H(c)$ can be set to 1. In both cases, (2.20) is satisfied as equality. As an example, the inequality induced by the 7− wheel of Figure 2.3 is:

$$3 \cdot x_{nnnn} + x_{nnk_0l_0} + x_{nnk_0l_1} + x_{i_0nnl_1} + x_{i_0nk_1n} + x_{i_1nk_2n} + x_{nj_0k_2n} + x_{nj_0nl_0} \leq 3$$

The wheels of G_A can be categorised with respect to three characteristics: the size p, the number of pairs of conjugate nodes D_p and the number of triple links on the rim T_p. These aspects affect both the configuration of the wheel subgraph and the form of the resulting (lifted) inequality. The following section examines the structural properties of the wheels of G_A.

2.5.1 Properties of wheels

Proposition 2.11 *Let $c_s, c_t \in H(c), s \neq t$, such that $(c, c_s), (c, c_t) \in M_1 \times M_2$, $M_1, M_2 = I, J, K, L$. Then $c \cap c_s = c \cap c_t$.*

Proof. This proposition states that at most one member of each double ground set can appear on the spokes. Let $M_1 = I$, $M_2 = J$ and $c_s = (n,n,k_0,l_0)$, $k_0, l_0 \neq n$, i.e. $c \cap c_s = (n,n) \in I \times J$. Assume $c_t \in H(c)$, such that $c \cap c_t \in I \times J$. It is obvious that $c \cap c_t = (n,n)$, i.e. $c \cap c_s = c \cap c_t$. ■

Proposition 2.12 *There can be at most two spokes based on the same double ground set. If exactly two spokes are based on the same double ground set, these spokes have to be rim adjacent.*

Proof. Assume w.l.o.g. that $c_s = (n,n,k_0,l_0), c_t = (n,n,k_1,l_1)$, $k_0, l_0, k_1, l_1 \neq n$. Then the spokes $(c, c_s), (c, c_t)$ are based on the double ground set $I \times J$ and $(c_s, c_t) \in E_C$. Unless nodes c_1, c_2 are rim adjacent, the edge (c_1, c_2) is a chord. If there exists another $c_u \in H(c)$, such that $(c, c_u) \in I \times J$, then one of the edges (c_1, c_u), (c_2, c_u), (c_1, c_2) will again be a chord. ■

Theorem 2.13 $p \leq 5$.

Proof. There exist six distinct double ground sets and, according to Prop. 2.12, at most two spokes can be based on each one. Therefore, the largest odd number of nodes in the rim is 11. ■

Proposition 2.14 *Let $c_s, c_t \in H(c)$, $s \neq t$ and assume $(c, c_s) \in M_1 \times M_2$, $(c, c_t) \in M_3 \times M_4$, $M_1, M_2, M_3, M_4 = I, J, K, L$. If $(c_s, c_t) \in E_c$, then, at least one of M_1, M_2 must coincide to either M_3 or M_4.*

Proof. This proposition states that two rim adjacent spokes must have at least one ground set in common. Assume that none of M_1, M_2 coincides with any of M_3, M_4. Let w.l.o.g. $M_1 \times M_2 = I \times J$, $M_3 \times M_4 = K \times L$, and $c_1 = (n, n, k_0, l_0), c_2 = (i_0, j_0, n, n)$, $i_0, j_0, k_0, l_0 \neq n$. It is obvious that (c_1, c_2) $\notin E_c$, which is a contradiction. ■

Proposition 2.15 *Let $(c, c_s) \in M_1 \times M_2, (c, c_t) \in M_2 \times M_3$ be a pair of rim adjacent spokes. If $M_1 = M_3$, then $(c, c_i) \subseteq (c_s, c_t), i = s, t$; otherwise,$(c_s, c_t) \in M_2 \times M_4$, where $M_4 \neq M_1, M_2, M_3$.*

Proof. This proposition reveals a mechanism for determining the double set of a rim edge, given the double sets of the adjacent spokes.

Assume w.l.o.g. that $M_1 = I, M_2 = J$. If $M_3 = M_1$, then let $c_s = (n, n, k_0, l_0), c_t = (n, n, k_1, l_1)$, $k_0, l_0, k_1, l_1 \neq n$. If either $k_0 = k_1$ or $l_0 = l_1$, then $(c_1, c_2) = (n, n, k_0)$ or $(c_1, c_2) = (n, n, l_0)$, respectively. In both cases, $(c, c_i) \subseteq (c_s, c_t), i = s, t$.

If $M_3 \neq M_1$ assume w.l.o.g. that $M_3 = K$, which implies $M_4 = L$. Let $c_1 = (n, n, k_0, l_0), c_2 = (i_1, n, n, l_1)$, $k_0, l_0, i_1, l_1 \neq n$. Then $|c_1 \cap c_2| > 1$, if and only if $l_0 = l_1$. But then $(c_1, c_2) = (n, l_0) \in J \times L$. ■

Example 2.1 *With respect to Figure 2.3, observe that the spokes for nodes $c_1 = (n, n, k_0, l_0)$, $c_2 = (n, n, k_0, l_1)$ are the only ones based on the double set $I \times J$. Notice that $(c_1, c_2) = (n, n, k_0) \in I \times J \times K$ and $(c, c_1) \subset (c_1, c_2)$. Note also that any pair of rim adjacent spokes have at least one set in common.*

Corollary 2.16 *For two rim adjacent nodes $c_s, c_t \in H(c)$, $s \neq t$, (c_s, c_t) is a triple link only if $c \cap c_s = c \cap c_t$.*

Proof. Assume $c \cap c_s \neq c \cap c_t$ and let w.l.o.g. $c_s = (n, n, k_0, l_0), c_t = (n, j_0, n, l_0)$. Since spokes are assumed to be based only on double ground sets, $j_0, k_0 \neq n$, which implies that (c_s, c_t) can never be a triple link. ■

Proposition 2.17 *Let (c, c_s), (c, c_t), (c, c_u) be rim adjacent spokes, no two of which are based on the same double ground set. Then all four sets must appear on the double sets of the spokes.*

Proof. Given that (c, c_s), (c, c_t), (c, c_u) are rim adjacent, assume that only three sets appear in the three rim adjacent spokes. W.l.o.g. and according to Prop. 2.14, assume $(c, c_s) \in I \times J$, $(c, c_t) \in J \times K$, $(c, c_u) \in I \times K$. By Prop. 2.15, it must be that $(c_s, c_t) \in J \times L$ and $(c_t, c_u) \in K \times L$. This implies that nodes c_s, c_u have index l in common. But both of these nodes have also index i equal to n. It follows that $(c_s, c_u) \in I \times L$, i.e. there is a chord. ■

Example 2.1 (cont.) *Examine, in Figure 2.3, the spokes corresponding to the three rim adjacent nodes (n, n, k_0, l_1), (i_0, n, n, l_1) and (i_0, n, k_1, n) and observe that all four sets appear on these three spokes.*

Theorem 2.18 *If D_p denotes the number of pairs of conjugate nodes:*
$p - 1 \leq D_p \leq p$, for $2 \leq p \leq 4$
$D_p = p$, for $p = 5$

Proof. It is easy to see that the upper bound is valid, since we cannot have more than p pairs of conjugate nodes in a rim of size $2p + 1$. Concerning the lower bound, the cases for each possible value of p are examined separately.

$p = 2$: If $D_2 = 0$, then we must have 5 spokes based on 5 distinct double ground sets. Assume w.l.o.g. that one spoke is based on $I \times J$ and that one of its rim adjacent spokes is based on $J \times K$, i.e. assume the sequence $[I \times J] - [J \times K]$. According to Prop. 2.17, the next spoke in the sequence must contain set L. The two possible options are double sets $J \times L$ and $K \times L$.

$[I \times J] - [J \times K] - [J \times L]$: The fourth spoke must contain set I, again because of Prop. 2.17. The two options are sets $I \times J$ and $I \times L$. Using $I \times J$ is forbidden by Prop. 2.12. Using set $I \times L$ results in the sequence $[I \times J] - [J \times K] - [J \times L] - [I \times L]$. In order for this sequence to be completed to a cyclic one, the only possible set is $J \times L$. However, using this set would create a chord.

$[I \times J] - [J \times K] - [K \times L]$: Because of Prop. 2.17, the fourth spoke must contain set I. Again set $I \times L$ is the only option. As shown above, the sequence $[I \times J] - [J \times K] - [J \times L] - [I \times L]$ cannot be extended.

It follows that $D_2 \geq 1$.

$p = 3$: $D_3 = 0$ implies that the wheel has 7 spokes based on distinct double ground sets. Since the number of distinct double ground sets is 6, this is impossible. If $D_3 = 1$, assume w.l.o.g. that the pair of spokes corresponding to the single pair of conjugate nodes is based on the double ground set $I \times J$. This implies a sequence of spokes $[I \times J] - [I \times J]$. The sets appearing on the rim adjacent spokes of this sequence must be selected according to one of the following exhaustive options:

(i) only one of the sets I, J and both sets K,L;

(ii) both sets I,J and only one of the sets K,L;

(iii) both sets I,J and both sets K,L.

Each case is treated separately.

(i) Let w.l.o.g. the sequence be extended as $[I \times K] - [I \times J] - [I \times J] - [I \times L]$. Consider the last three spokes and observe that the next spoke to the right must contain set K, according to Prop. 2.17. Such a spoke cannot contain set I because a chord would be then created. Therefore, the only option is a spoke based on set $K \times L$. The same reasoning proves that the next spoke to the left of the above sequence must also be based on set $K \times L$. The resulting sequence $[K \times L] - [I \times K] - [I \times J] - [I \times J] - [I \times L] - [K \times L]$ includes a chord.

(ii) Let w.l.o.g. the extended sequence be $[I \times K] - [I \times J] - [I \times J] - [J \times K]$. The next spokes to both the right and the left of this sequence must contain set L, according to Prop. 2.17. The choice is between the double sets $I \times L$, $K \times L$ for the next spoke to the left and between the double sets $J \times L$, $K \times L$ for the next spoke to the right, i.e. three possible extensions, since set $K \times L$ cannot be used for both. Assume the case of using set $K \times L$ for extending the sequence to the left and set $J \times L$ for extending the sequence to the right. The only set left for the seventh spoke is set $I \times L$. The resulting cyclic sequence is $[K \times L] - [I \times K] - [I \times J] - [I \times J] - [J \times K] - [J \times L] - [I \times L]$. The rim adjacent spokes $[I \times L] - [K \times L] - [I \times K]$ violate the condition of Prop. 2.17, i.e. a chord is formed.

It is always the case that two of the sets $I \times L$, $J \times L, K \times L$ appear at the spokes left and right of the above sequence and the last one will be used for the seventh spoke. It can be checked that there are always three consecutive spokes, which violate the condition of Prop. 2.17.

(iii) Let w.l.o.g. the augmented sequence be $[I \times K] - [I \times J] - [I \times J] - [J \times L]$. Based on Prop. 2.14 and Prop. 2.17, it is easy to see that the possible sets for the next spoke to the left of this sequence are sets $I \times L$ and $K \times L$, whereas the possible sets for the next spoke to the right are sets $J \times L$ and $K \times L$. This again gives rise to three cases, since using set $K \times L$ for both spokes would create a chord. Assume the case of the next spoke to the left being based on set $I \times L$ and the next spoke to the right being based on set $K \times L$. This results in the sequence $[I \times L] - [I \times K] - [I \times J] - [I \times J] - [J \times L] - [K \times L]$. According to Prop. 2.17, the next (and last) spoke to the right of this sequence must contain set I. If the second set is K, a chord is formed. If the second set is L, a second pair of conjugate nodes based on set $I \times L$ appears, in contradiction to the hypothesis of $D_3 = 1$. A similar contradiction, or a chord creation, is also the outcome of the other two possible extensions.

It follows that there exists no 7-wheel with a single pair of conjugate nodes. Therefore, it must be $D_3 \geq 2$, as required.

$p = 4$: If $D_4 = 0, 1, 2$, more that 6 spokes must be based on distinct double ground sets. Since this is impossible, it follows that $D_4 \geq 3$.

$p = 5$: As above, having less than 5 pairs of conjugate nodes would require the existence of at least 7 double ground sets for an equal number of spokes. This proves that $D_5 = 5$.The proof is complete.

■

Let us now focus on the number of triple links on the rim.

Proposition 2.19 *Consider four consecutive rim adjacent spokes* $(c, c_1), (c, c_2), (c, c_3), (c, c_4)$*, where nodes* c_2 *and* c_3 *are conjugate. Then* (c_2, c_3) *can be a triple link only if all four sets appear on the double sets of the four spokes.*

Proof. Assume that all four sets I, J, K, L appear on the spokes. Since the four spokes are rim adjacent and (c, c_2), (c, c_3) are based on the same double ground set, let w.l.o.g. $(c, c_1) \in I \times J$, $(c, c_2) \in J \times K, (c, c_3) \in J \times K$, $(c, c_4) \in K \times L$, based on Prop. 2.14. According to Prop. 2.15, if all rim edges are double links, $(c_1, c_2) \in J \times L$, $(c_2, c_3) \in J \times K$ and $(c_3, c_4) \in I \times K$. Notice that both c_1, c_3 have index j equal to n and both c_2, c_4 have index k equal to n.

If (c_2, c_3) was a triple link, then it would be either $(c_2, c_3) \in I \times J \times K$ or $(c_2, c_3) \in J \times K \times L$. If $(c_2, c_3) \in I \times J \times K$ then nodes c_2 and c_4 would also have index i in common, i.e. (c_2, c_4) would be a chord. Similarly, if $(c_2, c_3) \in J \times K \times L$, then (c_1, c_3) would be a chord. It follows that (c_2, c_3) cannot be a triple link, if all four sets appear on the spokes.

If only three sets appear on the spokes, assume w.l.o.g. that $(c, c_1) \in I \times J$, $(c, c_2) \in J \times K, (c, c_3) \in J \times K$, $(c, c_4) \in I \times K$ and, based on Prop. 2.15, $(c_1, c_2) \in J \times L$, $(c_2, c_3) \in J \times K$ and $(c_3, c_4) \in K \times L$. Notice that it can be $(c_2, c_3) \in I \times J \times K$ without a chord being formed between nodes c_2 and c_4. ■

Example 2.1 (cont.) *Consider the two quadruples of rim adjacent nodes:*

(a) $(n, j_0, n, l_0), (n, n, k_0, l_0), (n, n, k_0, l_1), (i_0, n, n, l_1)$;

(b) $(i_0, n, n, l_1), (i_0, n, k_1, n), (i_1, n, k_2, n), (n, j_0, k_2, n)$;

Observe the existence of a triple link within the first quadruple and the impossibility of a triple link within the second one.

Proposition 2.20 *If two pairs of conjugate nodes are consecutive, at most one triple link can appear at the corresponding edges.*

Proof. Assume w.l.o.g. the consecutive nodes c_1, c_2, c_3, c_4 and let the spokes of c_1 and c_2 be based on $I \times J$ and the spokes of c_3, c_4 be based on $J \times K$. Assume also that both (c_1, c_2) and (c_3, c_4) are triple links. The sequence of the four spokes is $[I \times J] - [I \times J] - [J \times K] - [J \times K]$. According to Prop. 2.19, the next spoke to the left of this sequence must be based on set $I \times K$ in order to allow for the existence of a triple link between c_1 and c_2. For the same reason, the next spoke to the left must also be based on set $I \times K$. But then, these two spokes must be adjacent, according to Prop. 2.12. Hence, the augmented sequence is cyclic and results in a wheel of even length. Therefore, a triple link can be established either between c_1 and c_2 or between c_3 and c_4. ■

Corollary 2.21 *If T_p denotes the number of triple links on the rim:*
$0 \leq T_p \leq p-1, p = 2, 3$
$0 \leq T_p \leq p-2, p = 4, 5$

Proof. According to Prop. 2.15, a triple link can exist only between two (adjacent) conjugate nodes. The number of pairs of conjugate nodes must therefore follow the conditions of Prop. 2.18. The lower bound of 0 on T_p is trivial, since, for sufficiently large n, one can always transform a triple link into a double one. The upper bound for T_p will be proved based on the conditions imposed by Prop. 2.18 and 2.20.

For $p = 2$, there can be either 1 or 2 pairs of conjugate nodes. If 2 pairs exist, these must be adjacent, hence at most one can include a triple link, according to Prop. 2.20. It follows that $T_2 \leq 1$.

For $p = 3$, 2 or 3 pairs of conjugate nodes can exist. If the rim includes two pairs of conjugate nodes, at most two triple links can appear on the rim. If three pairs exist, they will all be adjacent, hence a triple link can appear at most at 2 of them. In both cases, $D_3 \leq 2$, as required.

For $p = 4$, $D_4 = 3, 4$. For $D_4 = 4$, the four pairs of conjugate nodes are bound to be consecutive on the rim. It follows from Prop. 2.20 that at most two of the them can form triple links. Hence, $T_4 \leq 2$.

Given $D_5 = 5$, it is easy to see that the five pairs of conjugate nodes are consecutive, allowing for no more than 3 of them to form triple links on the rim. It follows that $T_5 \leq 3$. ■

Proposition 2.22 *There are $O(n^{20})$ wheels.*

Proof. Consider a wheel of size $2p + 1$. Any node of G_A can be the hub, hence there are n^4 options for the hub c. Given the hub, exactly two indices for each node in the rim are determined, since all spokes are assumed to be double links. The options for the other two indices depend on whether the node belongs to a pair of conjugate nodes or not. Assume w.l.o.g. that $c = (n, n, n, n)$ and let the rim be examined in a clockwise direction.

Observe that the number of indices from each set required for a wheel structure cannot be more that 7. For example, an 11−wheel must have all 6 double sets on the spokes. Consider w.l.o.g. set

I. All three double sets $I \times J$, $I \times K$, $I \times L$ appear on the spokes and at least two of them appear twice. Therefore, at least 5 nodes of the rim have value n for index i. This leaves 6 nodes with index i different from n. As a result, at most 7 values of index i are required for an 11−wheel. The argument can be repeated for each of the indices j, k, l and also for wheels of smaller size. There are n options for selecting the value of index i for the hub; $n-1$ options for selecting the value of index i for first node examined in the rim, which has index i different from n; $n-2$ for the second node examined and finally $n-6$ options for selecting the value of index i for the last node. It follows that there are $O(n)$ options for selecting each value of index i, the same being true for all other indices.

Two adjacent nodes, which do not form a pair of conjugates, are bound to have one index in common, which is different from n. This holds because the two nodes must be connected and is a direct consequence of Prop. 2.14. The nodes of the wheel in Figure 2.3 can be checked for an illustration. Hence, each node not followed by a conjugate node has $O(n)$ options for only one of its indices. This is the case for each node not belonging to a pair of conjugate nodes.

Two conjugate nodes are bound to be adjacent (Prop. 2.14). The first of these nodes has $O(n)$ options for each of its two indices, which are not equal to n, if there is not a triple link with its successor; otherwise, it has $O(n)$ options for only one of its indices. By restricting the analysis to wheels with no triple links, such a node has $O(n) \cdot O(n)$ options for its indices. Its successor, on the other hand, has again $O(n)$ options for only one index different from n, since its successor cannot be a conjugate node. It follows that the are $[O(n)]^3$ options for each pair of conjugate nodes.

There are D_p such pairs and $2p+1-2D_p$ nodes having no conjugate. Therefore, the number of rims of size $2p+1$, which include D_p pairs of conjugate nodes is:

$$[O(n)]^{3 \cdot D_p} \cdot [O(n)]^{2p+1-2 \cdot D_p} = [O(n)]^{2p+1+D_p} = O(n^{2p+1+D_p})$$

Since there are also n^4 options for the hub, the number of wheels of size $2p+1$ is:

$$O(n^{2p+1+D_p}) \cdot n^4 = O(n^{2p+D_p+5}).$$

Given that $D_p \leq p$, the function $2p + D_p + 5$ is strictly increasing with respect to p. For 11−wheels, $p = 5$ and $D_p = 5$. Hence, their number dominates the number of wheels of smaller size. Substituting the values for p, D_p in the above expression states that the number of 11− wheels is $O(n^{20})$. ■

This concludes the structural description of the wheels of G_A. The topic addressed in the next section is the form of the inequalities derived from this family of subgraphs.

2.5.2 Lifted wheel inequalities

As mentioned above, each wheel induces an inequality of the form:

$$p \cdot x_c + \sum_{d \in H(c)} x_d \leq p \tag{2.21}$$

or

$$3 \cdot x_{nnnn} + x_{nnk_0l_0} + x_{nnk_0l_1} + x_{i_0nnl_1} + x_{i_0nk_1n} + x_{i_1nk_2n} + x_{nj_0k_2n} + x_{nj_0nl_0} \leq 3 \tag{2.22}$$

for the 7− wheel of Figure 2.3.

Lifting inequality (2.21) requires including all variables $x_s \in C \backslash W(c)$ in the l.h.s., with the maximum possible coefficient, without raising the r.h.s. If a_s is the coefficient for $x_s \in C \backslash W(c)$, the augmented inequality has the form:

$$a_s \cdot x_s + p \cdot x_c + \sum_{d \in H(c)} x_d \leq p \tag{2.23}$$

If $x_s = 0$, (2.23) is reduced to (2.21). If $x_s = 1$, the inequality

$$\sum_{d \in H(c)} x_d \leq p - a_s \tag{2.24}$$

must hold. If node s is not connected to the hub c, x_c can still be set to 1, in which case (2.23) becomes $p \leq p - a_s$ or $a_s \leq 0$. Hence, only nodes connected to the hub can have a non-zero coefficient in the lifted inequality. Assuming $c = (n, n, n, n)$ for clarity, it is equivalent to state that only nodes having two or three of their indices equal to n should be considered.

Proposition 2.23 *If a_s is the coefficient of a variable x_s in a lifted wheel inequality, then:*
$a_s = p$, *if* $s \equiv d$;
$a_s \leq 2$, *if* $|c \cap s| = 3$;
$a_s \leq 1$, *if* $|c \cap s| = 2$.

Proof. The maximum value for a_s, i.e. p, is achieved only if s is connected to all nodes $s \in W(c)$, since, for $x_s = 1$, the l.h.s. of (2.24) would become 0. Obviously, this is possible only if $s = c$. Given also that $p \leq 5$, it follows that the maximum coefficient on the l.h.s. of a lifted wheel inequality is 5.

Consider a variable x_s, such that $s \in C \backslash W(c)$ and $|c \cap s| = 3$. Let w.l.o.g. $s = (i, n, n, n)$, $i \in I \backslash \{n\}$. Node s will be connected to the hub and also to all nodes of the rim, whose spokes are based on one of the double sets $J \times K$, $J \times L$, $K \times L$. There can be at most 6 such nodes, according to Prop. 2.12. Setting $x_s = 1$, implies that the hub together up to 6 nodes will be set to 0. This leaves $2p + 1 - 6 = 2 \cdot (p - 2) - 1$ nodes of the rim intact. Independently of whether these nodes are consecutive or not, at least $p - 2$ of them can be still be set to 1. Hence, (2.21) becomes $a_s + p - 2 \leq p$ or $a_s \leq 2$. The possible values for coefficient a_s are:

(i) $a_s = 0$, if node s is connected to 0 or 1 or 2 nodes of the rim;

(ii) $a_s = 1$, if node s is connected to 3 or 4 nodes of the rim;

(iii) $a_s = 2$, if node s is connected to 5 or 6 nodes of the rim;

It is easy to see that the coefficient a_s is identical for all variables $\{x_{innn} : i \in I \backslash \{n\}\}$.

Now, consider a variable x_s, such that $s \in C \backslash W(c)$, $|c \cap s| = 2$ and let w.l.o.g. $s = (i, j, n, n)$, $i \in I \backslash \{n\}$, $j \in J \backslash \{n\}$. Node s will be connected to the hub and also to all nodes of the rim, whose spokes are based on the double set $K \times L$. There can be at most 2 such nodes, according to Prop. 2.12. Setting $x_s = 1$, forces the hub and at most 2 adjacent nodes of the rim to take value 0. At least $2p - 1$ nodes are left intact and p of them can be set to 1. Using (2.23), it follows that $a_s \leq 0$. The coefficient of x_s will be non-zero only if s is connected to 3 adjacent nodes of the rim or 2 non-adjacent pairs of adjacent nodes. This is the case only if both of the following conditions hold.

Condition I The wheel includes a pair of conjugate nodes, whose spokes are based on the double set $K \times L$.

Condition II Either i or j has a value i_0 or j_0, respectively, which is common to two adjacent non-conjugate nodes. These two nodes also have index k or l equal to n.

Note that if the nodes were conjugate in Cond.II, the rim edge connecting them would be a triple link. As a result, node s could be connected to this pair of conjugate nodes only, which would imply $a_s \leq 0$. Assuming w.l.o.g. that index i_0 does satisfy Cond.II, all variables $\{x_{i_0jnn} : j \in J\backslash\{n\}\}$ would be included in the inequality with a coefficient equal to 1. ■

Example 2.1 (cont.) *The nodes/variables included in the lifting of the wheel in Figure 2.3 are depicted at Figure 2.4.*

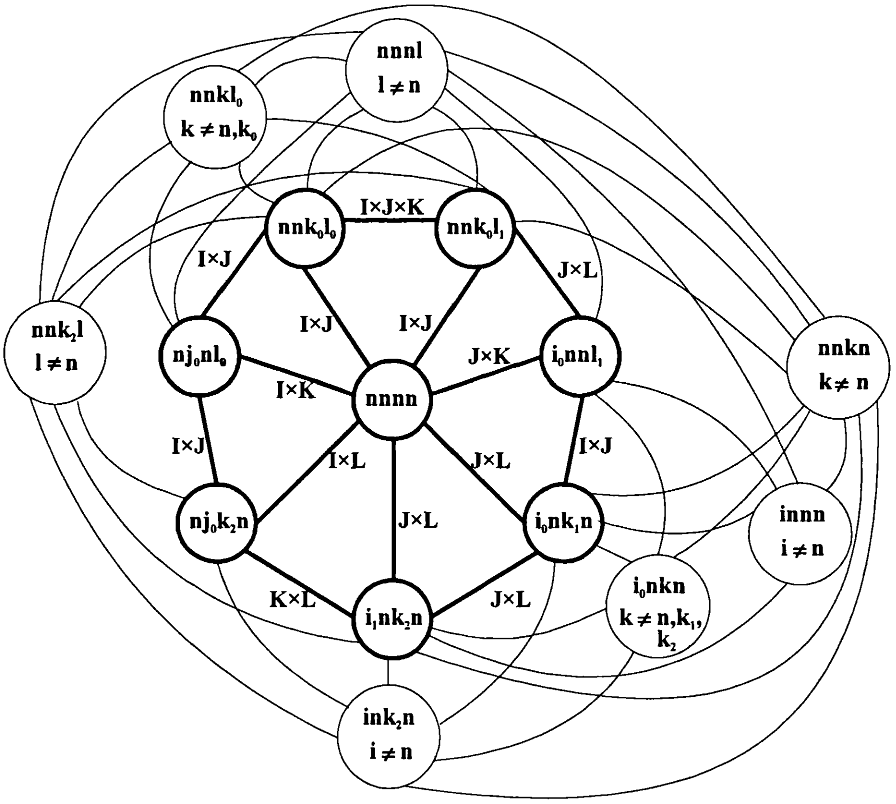


Figure 2.4: The nodes of a lifted wheel inequality

Nodes $\{nnkn : k \neq n\}$ are connected to four nodes of the rim. Setting any of these variables to 1*, does not affect* 3 *variables of rim,* 2 *of which can be set to* 1*. Nodes $\{innn : i \neq n\}$ and $\{nnnl : l \neq n\}$ are connected to* 3 *nodes of the rim. Setting any of these to* 1 *leaves intact* 4 *nodes of rim,* 2 *of which can be set to* 1*. Therefore, $a_{innn} = a_{nknn} = a_{nnnl} = 1$. Since nodes $\{njnn : j \neq n\}$ are connected to only* 2 *nodes of the rim, $a_{njnn} = 0$.*

Concerning nodes having 2 *indices in common with the hub, observe that the two pairs of conjugate nodes are based on the double sets $I \times J$ and $J \times L$. Hence, only nodes of the form nnkl and inkn are examined. Values k_2, l_0 satisfy cond.II, hence $a_{nnk_2l} = a_{nnkl_0} = 1$. Value k_0 does appear in two nodes of the rim, but is within a triple link, therefore $a_{nnk_0l} = 0$. Similarly, since value i_0 satisfies cond.II, $a_{i_0nkn} = a_{ink_2n} = 1$.*

The lifted wheel inequality is:

$$\begin{gathered}3 \cdot x_{nnnn} + x_{nnk_0l_0} + x_{nnk_0l_1} + x_{i_0nnl_1} + x_{i_0nk_1n} + x_{i_1nk_2n} + x_{nj_0k_2n} + x_{nj_0nl_0} + \\ \sum_{i \in I \backslash \{n\}} x_{innn} + \sum_{k \in K \backslash (n\}} x_{nnkn} + \sum_{l \in L \backslash \{n\}} x_{nnnl} + \\ \sum_{i \in I \backslash \{n,i_1\}} x_{ink_2n} + \sum_{k \in K \backslash \{n,k_0\}} x_{nnkl_0} + \sum_{k \in K \backslash \{n,k_1,k_2\}} x_{i_0nkn} + \sum_{l \in L \backslash \{n\}} x_{nnk_2l} \leq 2\end{gathered}$$

The number of coefficients in the inequality above is independent of the lifting sequence. It is easy to verify that the procedure of devising the lifted inequality includes a constant number of steps. There are 4 sets of variables having 3 indices in common with the hub. Similarly, there are 6 possibilities for a variable to have 2 of its indices equal to n. For each one of the remaining 2 indices, the options of having a value, which is also common to a pair of adjacent, non-conjugate nodes, are trivially bounded by the value of p. Recall that $p \leq 5$.

The number of lifted wheel inequalities is at least of the same order as the number of wheels of G_A. These inequalities have a r.h.s. of at most 5, and coefficients for the l.h.s. with values up to 5 for the hub and up to 2 for all other variables. The number of variables in the l.h.s. is $O(n)$. The lifting procedure results in maximal inequalities, i.e. inequalities that cannot be further strengthened.

2.6 A family of composite cliques of G_A

This section exhibits a class of inequalities, derived from G_A, which can be regarded either as lifted wheel inequalities or arising from a subgraph of G_A formed by a composition of cliques.

Let $c, d \in C$ such that $|c \cap d| = 0$. Define:

$$\begin{aligned} Q^1(c) &= \{p \in C : |c \cap p| = 3\} \\ Q^2(c,d) &= \{p \in C : |c \cap p| = |d \cap p| = 2\} \end{aligned}$$

and $W(c,d) = Q^1(c) \cup Q^2(c,d)$.

Consider the inequality

$$2x_c + \sum \{x_q : q \in W(c,d)\} \leq 2 \tag{2.25}$$

To show the validity of (2.25), observe that, for $c = (n,n,n,n), d = (i_0, j_0, k_0, l_0)$, inequality (2.25) becomes:

$$\begin{aligned} 2x_{nnnn} + \sum_{i \neq n} x_{innn} + \sum_{j \neq n} x_{njnn} + \sum_{k \neq n} x_{nnkn} + \sum_{l \neq n} x_{nnnl} & \\ + x_{i_0j_0nn} + x_{i_0nk_0n} + x_{nj_0k_0n} + x_{i_0nnl_0} + x_{nj_0nl_0} + x_{nnk_0l_0} & \leq 2 \end{aligned} \tag{2.26}$$

This inequality can be derived by lifting the wheel inequality induced by the 5-wheel of Figure 2.5. Notice that this wheel belongs to the class of wheels with triple sets on the spokes, which was not examined in the previous section.

Alternatively, notice that the node set $c \cup Q_1(c)$ forms a clique of type II (Section 2.3). Notice also that nodes $\{(n,n,n,n), (i_0, j_0, n, n), (i_0, n, k_0, n), (i_0, n, n, l_0)\}$ form a clique of type III and that three more cliques of these type are formed among c and the six variables of $Q_2(c)$. In this sense, the subgraph $W(c,d)$ can be considered as a composition of cliques. Finally, notice that each $d \in Q_2(c)$

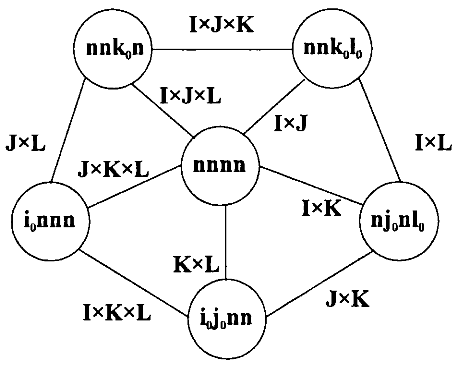


Figure 2.5: A 5-wheel with triple sets on the spokes

is connected to all but one nodes in $Q_2(c)$. The variables of the subgraph $W(c,d)$ can be set to one in three different patterns:

(i) $x_c = 1$, implying all other variables are set to 0

(ii) $x_d = x_f = 1$ where $d \in Q_1(c), f \in Q_2(c)$ and $|d \cap f| \in \{0,1\}$. In this case, all variables in $c \cup Q_1(c)$ are set to 0 and the node in $Q_2(c)$ not connected to f is always connected to d.

(iii) $x_d = x_f = 1$ where $d, f \in Q_2(c)$ and $|d \cap f| \in \{0,1\}$. By inspection, it can be seen that all variables in $c \cup Q_1(c)$, together with the remaining 4 variables in $Q_2(c)$, are set to 0.

This proves the validity of inequality (2.26) and consequently (2.25).

Proposition 2.24 *Inequality (2.25) is maximal.*

Proof. Suppose that there exists a variable x_p ($p = (i_p, j_p, k_p, l_p) \in C$) which can be added to the left-hand side of (2.25) without increasing the r.h.s. Clearly, x_p must have two indices in common with c, since otherwise it can be set to 1 together with x_{nnnn} (all variables with three indices in common with c have already been included). Assume then c, d as above and let $i_p = n, j_p = n$. Observe that variables $x_{nnk_pl_p}$ and $x_{nj_0nl_0}, x_{i_0nnn}$ can be set simultaneously to 1. This is valid even if $k_p = k_0$, $l_p \neq l_0$ or $l_p = l_0$, $k_p \neq k_0$.

It is also easy to see that no coefficient can be increased without raising the r.h.s. of (2.25). ■

By applying an argument similar to that of Prop. 2.10, it can be seen that the number of distinct inequalities (2.25) is $\frac{1}{16}n^4 \cdot (n-1)^4$.

2.7 The Odd holes of G_A

An odd hole is defined as a chordless hole of odd length. Since two nodes of G_A are connected if and only if they have 2 or 3 indices in common, the formal definition of an odd hole in G_A is the following.

Definition 2.3 *Let $p \geq 2$ and integer. A node set $H \subseteq C$ induces an odd hole if and only if it can be ordered into a sequence $\{c_0, ..., c_{2p}\}$, such that:*

$$|c_s \cap c_t| = \left\{ \begin{array}{ll} 2 \text{ or } 3, & \text{if } t = s \pm 1 \mod(2p+1) \\ 0 \text{ or } 1, & \text{otherwise} \end{array} \right\}$$

for all $c_s, c_t \in H$, $s \neq t$.

Each odd hole H induces an inequality of the form:

$$\sum_{d \in H} x_d \le p \tag{2.27}$$

Note that wheels are essentially special cases of odd holes, with an extra node (hub) connected to all $2p+1$ nodes.

This section exhibits a number of properties regarding the odd holes of G_A. Two classes of odd holes are also presented. The first of them, called odd holes of type I, is derived from the odd holes of the $3AAP_n$ problem, introduced in [9]. The second, called odd holes of type II, is derived from a new class of odd holes for the $3PAP_n$ problem. Each class is further analysed and the form of lifted odd hole inequalities is also presented.

Recall first that the edge between two adjacent nodes in the odd hole can be either double or triple, as defined in Section 2.2.

Proposition 2.25 *Two adjacent edges of an odd hole cannot be based on the same double ground set.*

Proof. Assume w.l.o.g. edges (c_1, c_2), (c_2, c_3). Let $(c_1, c_2) \in I \times J$ and $c_1 = (i_1, j_1, k_1, l_1)$, $c_2 = (i_1, j_1, k_2, l_2)$, $k_1 \neq k_2$, $l_1 \neq l_2$. If (c_2, c_3) is also based also on $I \times J$ or on a triple ground set, which contains single sets I and J, node c_3 will have the form $c_3 = (i_1, j_1, k_3, l_3)$, $k_1 \neq k_2$ or $l_1 \neq l_2$. This implies that (c_1, c_3) will be a chord, a fact that contradicts to the definition of an odd hole. ■

Proposition 2.26 *Two adjacent edges cannot both be triple links.*

Proof. Assume w.l.o.g. an edge $(c_1, c_2) \in I \times J \times K$. An adjacent edge, which is also based on a triple ground set, is bound to contain two of the three sets I, J, K. This would imply the creation of a chord, which is a contradiction. ■

The following proposition is based on [9, Prop.5.3].

Proposition 2.27 *The number t of triple links in an odd hole of length $2p+1$ must satisfy*

$$max\left\{0, \left\lceil \frac{5}{2}(2p+1) - 3n^2 \right\rceil \right\} \le t \le p \tag{2.28}$$

Proof. The number of positive components of the vector $a_T = a^{c_1} + \ldots + a^{c_{2p+1}}$ is $5 \cdot (2p+1) - 2t$. This number cannot exceed the number of rows of matrix A, i.e. $6n^2$. The inequality $5 \cdot (2p+1) - 2t \le 6n^2$ gives the lower bound of (2.28). To prove the validity of the upper bound, assume an odd hole with $p+1$ triple links. This hole should have two triple links incident to the same node. This contradicts to Prop. 2.26. ■

Proposition 2.28 *The maximum coefficient in the l.h.s. of a lifted odd hole inequality is* 5.

Proof. Consider a variable x_s, $s \in C \backslash H$, and let a_s be its coefficient in a lifted odd hole inequality. The form of the inequality will be the following:

$$a_s x_s + \sum_{d \in H} x_d \le p \tag{2.29}$$

There can be at most 2 nodes of the hole, which have exactly the same indices in common with node s and, if there are exactly 2, they must also be adjacent. The argument supporting this point is exactly the same as the one used in Prop. 2.12. Since there are 6 double ground sets, there can be at most 6 pairs of adjacent nodes having exactly 2 indices in common with node s. Note that allowing s to have triple links with at least one node of the hole reduces the maximum number of connected nodes even further. Therefore, there can be at most 12 nodes of the hole connected to node s.

Setting $x_s = 1$, sets all these 12 nodes to 0. This leaves, for sufficiently large p, $2p + 1 - 12 = 2 \cdot (p - 5) - 1$ nodes of the odd hole unaffected. Independently of whether these nodes are consecutive or not, $p - 5$ of them can be set to 1. Inequality (2.29) becomes $a_s + p - 5 \leq p$ or $a_s \leq 5$. ■

The above proposition implies that wheels, although special cases of odd holes, give rise to lifted inequalities with the maximum possible coefficients in the l.h.s. In this sense, the particular structure exhibited by wheels allows for the derivation of inequalities equally complex to those arising from any possible odd hole.

2.7.1 Odd holes of type I

These odd holes are derived from the odd holes of the $3AAP_n$, as described in [9, Section 5]. Their construction is direct, arising from the observation that the edges of the intersection graph of $3AAP_n$ can be either single or double links.

Recall from Section 2.1 that the $3AAP_n$ is defined on three disjoint n-sets, namely I, J, K. Let H be the node set of an odd hole in the intersection graph of $3AAP_n$, i.e. $c \in I \times J \times K, \forall c \in H$. Adding $l = l_1$ to all $c \in H$ results in a node set $\widetilde{H} = \{(c, l_1) : c \in H\}$, which forms an odd hole in G_{A_n}. Note that $\widetilde{c} = (c, l_1) \in I \times J \times K \times L$ for all $\widetilde{c} \in \widetilde{H}$. The single and double links of H become double and triple links in $\widetilde{H}$, respectively. Based on this fact, it is trivial to extend all the results of [9] to this class of odd holes of G_A. Note that the distinct property of this class is that all nodes of the hole have identical values for a certain index. Equivalently, all odd holes of G_A, all nodes of which have a certain index in common, correspond to an odd hole in the intersection graph of the $3AAP_n$ problem. Odd holes having this property will be called as odd holes of type I. Such an odd hole is illustrated at Figure 2.6. Note that the index value(s) appearing on an edge is the one(s) being common to its incident nodes.

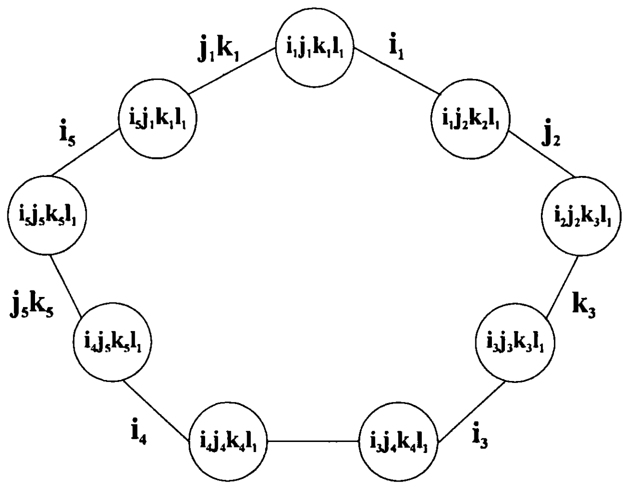


Figure 2.6: A 9-cycle of type I

To formally describe this class, select any 3 out of 4 single ground sets and a single value from the fourth set. Assume w.l.o.g. sets I, J, K and value $l_1 \in L$. Select subsets $Q_I \subset I$, $Q_J \subset J$, $Q_K \subset K$, satisfying $|Q_I| + |Q_J| + |Q_K| = 2p+1$, $1 \leq |Q_M| \leq p$ for $M = I, J, K$. Define the set $Q = Q_I \cup Q_J \cup Q_K$, $|Q| = 2p+1$.The associated odd hole $\widetilde{H}$ is the one where all nodes have index l equal to l_1 and edges are labelled by the members of set Q. Note that set Q, together with value l_1, defines also a row subset of matrix A, namely the rows indexed by $R^H = \{(q, l_1) : q \in Q\}$. The matrix formed by this row subset and the columns $c \in \widetilde{H}$ is identical to the circulant matrix C^2_{2p+1}, up to row and column permutations ([9, Prop.5.5]).

As an extension of the properties exhibited in [9], odd holes of type I, with length $2p+1$, exist for $p \leq n-1$. The number of triple links t must satisfy the inequality $\max\{0, 2(2p+1) - 3n\} \leq t \leq p-1$. Note that these bounds are tighter than the ones of inequality (2.28).

Consider the inequality:

$$\sum\{x_s : |s \cap (Q \cup \{l_1\})| \geq 3\} \leq p \tag{2.30}$$

For $p \leq n-1$, this inequality can be regarded as a lifted odd hole inequality. Its Chvâtal rank is 1, since it can be derived by adding all rows $\{(q, l_1) : q \in Q\}$, dividing the resulting inequality by 2, replacing $=$ by $\leq$ and rounding down both sides. If $|Q_I| = r$, $|Q_J| = s$, $|Q_K| = t$, the number of variables in the l.h.s. of (2.30) is $n \cdot (rs + rt + st) - 2rst$ ([9]).

The number of distinct lifted odd hole inequalities for the $3AAP_n$ is $O(2^{3n})$. Since each such inequality can be extended to an inequality of the $4PAP_n$ by considering any of the n values for the fourth index, it follows that the number of lifted odd holes inequalities of type I is $O(n \cdot 2^{3n})$.

2.7.2 Odd holes of type II

This class of odd holes appears both in the $3PAP_n$ and the $4PAP_n$ problem. Therefore, it can be better exhibited by examining first its appearance in $3PAP_n$.

Recall from Section 2.1 that the $3PAP_n$ is defined on three disjoint n-sets, namely I, J, K. Consider subsets of the three ground sets $Q'_I \subseteq I$, $Q'_J \subseteq J$, $Q'_K \subseteq K$, such that $|Q'_I| = |Q'_J| = |Q'_K| = m$. Assume any permutation of the elements of the three subsets $Q'_I = \{i_1, ..., i_m\}$, $Q'_J = \{j_1, ..., j_m\}$, $Q'_K = \{k_1, ..., k_m\}$, $3 \leq m \leq n$.

Proposition 2.29 *The node set*

$$Q' = \left\{ \begin{array}{ll} i_1j_1k_1,\ i_1j_1k_2,\ i_1j_2k_2,\ i_2j_2k_2, ..., & \\ i_mj_mk_m,\ i_mj_mk_1,\ i_mj_1k_1, & m \text{ odd} \\ i_1j_1k_1,\ i_1j_1k_2,\ i_1j_2k_2,\ i_2j_2k_2,\ ..., & \\ i_mj_mk_m,\ i_mj_qk_m,\ i_mj_qk_1,\ i_mj_1k_1, & m \text{ even},\ q \neq 1, m-1, m \end{array} \right\}$$

induces an odd hole in the intersection graph of $3PAP_n$.

Proof. Recall that in the intersection graph of both the $3PAP_n$ and the $4PAP_n$ two nodes are connected if they have at least two indices in common([37]). Considering the order of members of Q to be cyclic, each node is connected only to its predecessor and its successor according to this order.

The number of nodes is $3m$ for m odd and $3m+1$, if m is even. In both cases the length of the hole is odd. ■

For the purpose of illustrating this odd hole in the form of a Latin square, assume w.l.o.g. that $Q'_I = Q'_J = Q'_K = \{1, 2, ..., m\}$. The odd hole includes the nodes depicted at Table 2.1. Note that q can be equal to $2, 3, .., m-2$.

The inequality induced by such an odd cycle has the form:

Table 2.1: Odd cycle for m odd and m even

	1	2		m-1	m
1	1,2	2			
2		2,3	3		
			...		
m-1				m-1,m	m
m	1				m,1

	1	2		q	m-1	m
1	1,2	2				
2		2,3	3			
				...		
m-1					m-1,m	m
m	1			m,1		m

$$x_{i_1j_1k_1} + x_{i_1j_1k_2} + ... + x_{i_mj_mk_m} + x_{i_mj_mk_1} + x_{i_mj_1k_1} \leq p \tag{2.31}$$

if m is odd and

$$x_{i_1j_1k_2} + x_{i_1j_2k_2} + ... + x_{i_mj_mk_m} + x_{i_mj_qk_m} + x_{i_mj_qk_1} \leq p \tag{2.32}$$

if m is even.

Note that $p = \left\{ \begin{array}{ll} \frac{3m-1}{2}, & m \text{ odd} \\ \frac{3m}{2}, & m \text{ even} \end{array} \right\}$.

The extension of this class to $4PAP_n$ is quite straightforward. Consider $m-$subsets of the four $n-$sets, namely $Q_I \subseteq I$, $Q_J \subseteq J$, $Q_K \subseteq K$, $Q_L \subseteq L$. Assume any permutation of these four subsets and let $Q_I = \{i_1, ..., i_m\}$, $Q_J = \{j_1, ..., j_m\}$, $Q_K = \{k_1, ..., k_m\}$, $Q_L = \{l_1, ..., l_m\}$, $3 \leq m \leq n$. The node set of the associated odd hole has the form:

$$Q = \left\{ \begin{array}{ll} i_1j_1k_1l_1,\ i_1j_1k_2l_2,\ i_1j_2k_2l_3, ...,\ i_mj_mk_ml_{m-2},\ i_mj_mk_1l_{m-1},\ i_mj_1k_1l_m, & m \text{ odd} \\ i_1j_1k_1l_1,\ i_1j_1k_2l_2,\ i_1j_2k_2l_3, ...,\ i_{m-1}j_mk_ml_{m-3},\ i_mj_mk_ml_{m-2},\ i_mj_mk_1l_1, & m \text{ even} \end{array} \right\}$$

An odd hole subgraph for $m = 3$, is depicted at Figure 2.7. Figure 2.8 illustrates an odd hole for $m = 4$.

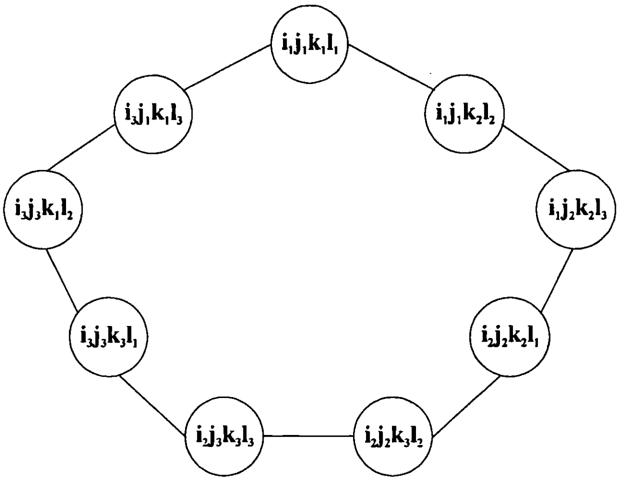


Figure 2.7: A 9-cycle of type II, (m=3)

Odd holes of this type exhibit a number of noticeable properties, namely:

(i) All edges belonging to the hole are double links.

(ii) Every three consecutive nodes have exactly one index in common.

(iii) There is a direct correspondence to an odd hole of the $3PAP_n$, since removing a particular index, e.g. index l in the example of Figure 2.7, results in an odd hole for $3PAP_n$. There are exactly $m!$ odd holes of $4PAP_n$, which can be derived from a particular odd hole of $3PAP_n$.

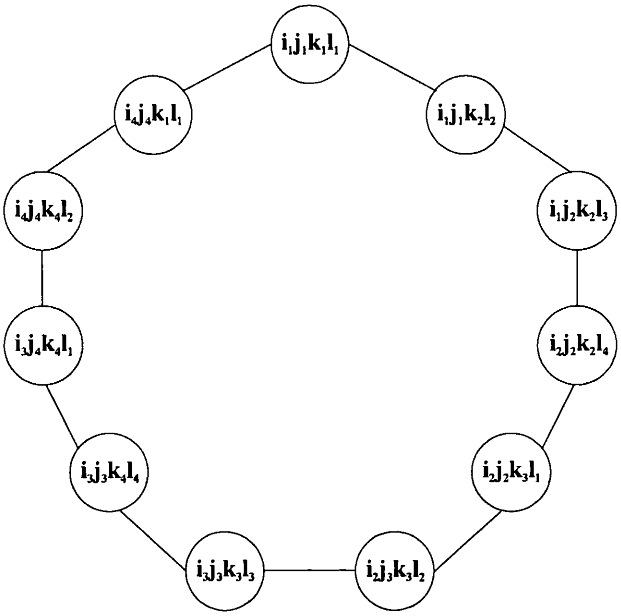


Figure 2.8: An 11-cycle of type II (m=4)

Table 2.2: Odd cycle of G_A, m odd

	1	2		m-1	m
1	11,22	23			
2		24,35	36		
			...		
m-1				(m-1)(m-3)	m(m-2)
m	1				m(m-1),1m

(iv) The single ground sets appearing at the links are only three, namely I, J, K in the hole of Figure 2.7. The double links follow the (cyclic) order $I \times J, I \times K, J \times K$. The only exception is the "last" link for the case of m being even, where the double set is $K \times L$.

To illustrate this type of odd hole in the form of a pair of OLS, assume $Q_I = Q_J = Q_K = Q_L = \{1, ..., m\}$, m odd. The odd hole is depicted at Table 2.2.

Lemma 2.30 *For odd holes of type II, $p \leq \lfloor \frac{3n-1}{2} \rfloor$.*

Proof. The cardinality of each subset $Q_S = Q \cap S$, $S = I, J, K$, cannot be more than n, i.e. $m \leq n$. If n is odd, the largest hole has length $3n$, and $p \leq \frac{3n-1}{2}$. If n is even, the largest hole has length $3n - 1$ and $p \leq \frac{3n-2}{2}$. In both cases, $p \leq \lfloor \frac{3n-1}{2} \rfloor$ ■

The inequality induced by an odd hole of size $2p + 1$ has the form:

$$X(Q) = \sum_{q \in Q} x_q \leq p \tag{2.33}$$

where $p = \left\{ \begin{array}{ll} \frac{3m-1}{2}, & m \text{ odd} \\ \frac{3m-2}{2}, & m \text{ even} \end{array} \right\}$.

This inequality can be further strengthened by applying the process of sequential lifting. Let

$x_s, s \in C\backslash Q$ and a_s the lifting coefficient. The augmented inequality would be:

$$a_s x_s + \sum_{q \in Q} x_q \leq p \tag{2.34}$$

If node s is connected to less than 3 nodes of the odd hole, setting $x_s = 1$ leaves intact at least $2p - 1$ nodes, p of which can be set to 1. In that case, ineq.(2.34) becomes $a_s + p \leq p$ or $a_s \leq 0$. Therefore, only nodes connected to at least 3 nodes in the hole are examined. Let m be odd, and consider the nodes $\{i_1 j_1 k_1 l : l \in L\backslash\{l_1\}\}$, $\{i_1 j_1 k_2 l : l \in L\backslash\{l_2\}\}, ..., \{i_m j_m k_1 l : l \in L\backslash\{l_{m-1}\}\}$, $\{i_m j_1 k_1 l : l \in L\backslash\{l_m\}\}$. Note that the first three indices are given by the nodes of the corresponding odd hole of the $3PAP_n$. It it easy to see that each such node is connected to exactly 3 consecutive nodes of the odd hole in hand. The lifting coefficient for each such node can be easily seen to be 1, independently of the lifting sequence. The resulting inequality has the form:

$$\begin{gathered} x_{i_1 j_1 k_1 l_1} + x_{i_1 j_1 k_2 l_2} + ... + x_{i_m j_m k_1 l_{m-1}} + x_{i_m j_1 k_1 l_m} + \\ \sum_{l \neq l_1} x_{i_1 j_1 k_1 l} + \sum_{l \neq l_2} x_{i_1 j_1 k_2 l} + ... + \sum_{l \neq l_{m-1}} x_{i_m j_m k_1 l} + \sum_{l \neq l_m} x_{i_m j_1 k_1 l} \leq p \end{gathered}$$

which is identical to

$$\sum_{l \in L} x_{i_1 j_1 k_1 l} + \sum_{l \in L} x_{i_1 j_1 k_2 l} + ... + \sum_{l \in L} x_{i_m j_m k_m l} + \sum_{l \in L} x_{i_m j_m k_1 l} + \sum_{l \in L} x_{i_m j_1 k_1 l} \leq p \tag{2.35}$$

The lifted odd hole inequality for m even has the form:

$$\sum_{l \in L} x_{i_1 j_1 k_2 l} + \sum_{l \in L} x_{i_1 j_2 k_2 l} + ... + \sum_{l \in L} x_{i_m j_m k_m l} \leq p \tag{2.36}$$

The l.h.s. of (2.35) has $3m^2$ variables and the l.h.s. of (2.35) has $3m \cdot (m-1) + 2$ variables. The nodes corresponding to the variables of (2.35), for $m = 3$, are illustrated at Figure 2.9.

Proposition 2.31 *The total number of inequalities (2.35), (2.36) is* $\sum_{m=3}^{n} \left[4 \cdot \binom{n}{m}^3 \cdot (m!)^3\right]$.

Proof. Each inequality uses m indices from three of the four ground sets and all n indices from the fourth one. Each of the three m−subsets can be selected in $\binom{n}{m}$ different ways, independently of the choice of the other two subsets. Moreover, there are $m!$ permutations of each subset, each one resulting in a distinct odd hole. Finally, the roles of the four sets are interchangeable, thus allowing 4 possibilities for selecting the three sets, which will provide the three m−subsets. ■

It is easy to see, using an analogous argument, that the number of inequalities (2.31), (2.32) is $\sum_{m=3}^{n} \left[\binom{n}{m}^3 \cdot (m!)^3\right]$.

2.8 Odd antiholes

An *odd antihole* $\overline{H}$ is defined as the complement of an odd hole subgraph ([18]).

Definition 2.4 *Let* $p \geq 2$ *and integer. A node set* $\overline{H} \subseteq C$ *induces an odd antihole of size* $2p + 1$ *if and only if it can be ordered into a sequence* $\{c_0, ..., c_{2p}\}$, *such that:*

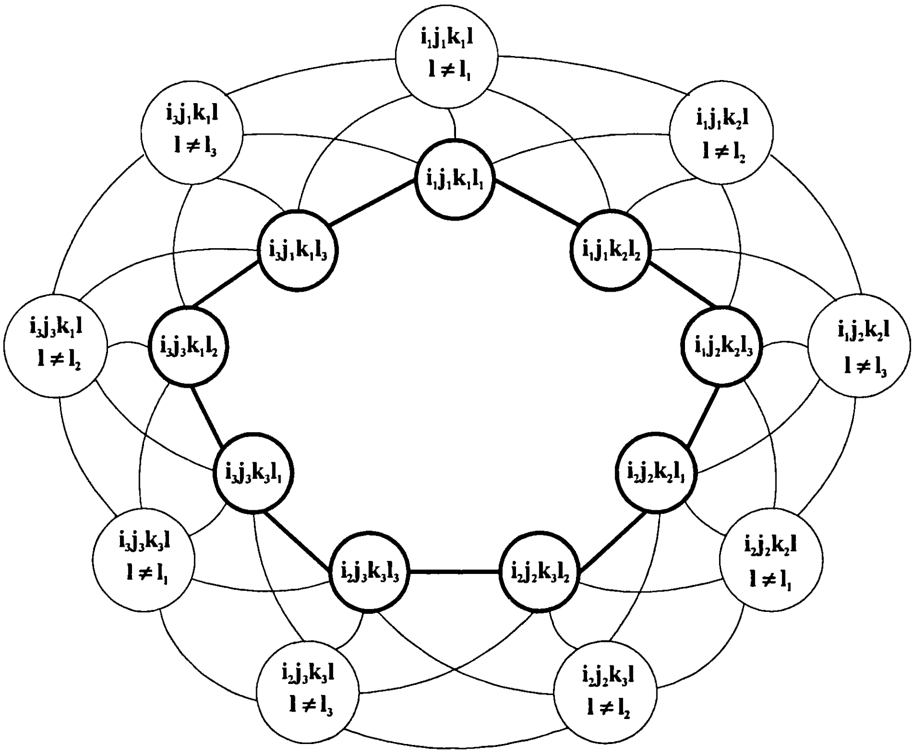


Figure 2.9: Nodes of a lifted 9-cycle inequality

$$|c_s \cap c_t| = \left\{ \begin{array}{ll} 0 \; or \; 1, & if \; t = s \pm 1 \mod(2p+1) \\ 2 \; or \; 3, & otherwise \end{array} \right\}$$
for all $c_s, c_t \in H$, $s \neq t$.

According to this ordering, each node of $\overline{H}$ is connected to all nodes of $\overline{H}$, except for its predecessor and successor. Therefore, the odd antiholes coincide with the circulants $C(2p+1, p)$ ([18]).

Each odd antihole $\overline{H}$ induces an inequality of the form:

$$\sum_{d \in H} x_d \leq 2 \tag{2.37}$$

This section exhibits a number of properties regarding the odd antiholes of G_A. Based on these properties, it is proved that odd antiholes of size larger than 5 do not exist in G_A. Note that a 5-antihole is also a 5-hole. Therefore, we assume $p \geq 3$ throughout this section.

Proposition 2.32 *For all* $c, d \in \overline{H}$, $|c \cap d| \neq 0$.

Proof. Assume w.l.o.g. $c = (i_0, j_0, k_0, l_0)$ and $d = (i_1, j_1, k_1, l_1)$, such that $|c \cap d| = 0$. According to definition 2.4, these nodes must be consecutive. Since $p \geq 3$, there are at least three consecutive nodes t_1, t_2, t_3, which are consecutive neither to c nor to d. It follows that all three nodes must be connected to both c and d and that $|t_1 \cap t_2| = |t_2 \cap t_3| \in \{0, 1\}$. Let w.l.o.g. $t_2 = (i_0, j_0, k_1, l_1)$. The only other node, which has at least two indices in common with both c and d and at most one index in common with t_2 is node (i_1, j_1, k_0, l_0). This implies that either $t_1 = t_3$ or there is an edge between t_2 and one of the nodes t_1, t_3. In both cases, there is a contradiction. Therefore, all nodes of an antihole must have at least one index in common. ■

Proposition 2.33 *All nodes of an antihole $\overline{H}$ have the same index in common.*

Proof. Assume a node $t = (i_t, j_t, k_t, l_t)$ and a pair of consecutive nodes c, d, none of which is the predecessor or the successor of t. It follows that t is connected to both c and d. According to Prop. 2.32, nodes c, d must have exactly one index in common. Let w.l.o.g. $c = (i_0, j_0, k_0, l_0)$ and $d = (i_0, j_1, k_1, l_1)$. If $i_t \neq i_0$, node t is bound to have either two indices in common with c and one index in common with d or the opposite. In both cases, t cannot be connected to both c and d, i.e. there is a contradiction. Therefore, it must be $i_t = i_0$. Repeating this argument for all nodes of $\overline{H}$ shows that all nodes must have index i equal to i_0. ∎

The following proposition is the most involved, therefore it is important to emphasise that assumptions at each step of the proof are made without loss of generality.

Proposition 2.34 *For $c, d \in \overline{H}$, $|c \cap d| \neq 3$.*

Proof. Assume w.l.o.g. $c = (1,1,1,1)$ and $d = (1,1,1,2)$. According to Definition 2.4, these nodes cannot be consecutive. Hence, the number of nodes between c and d can be 1 or more. We examine these cases separately. Note that, according to Prop. 2.33, all nodes of $\overline{H}$ are bound to have index i equal to 1.

Case 2.34.1 *There is exactly one node t_1 between c and d. We will illustrate that the sequence of nodes c, t_1, d is not extendible to an antihole. Figure 2.10 depicts the extended sequence.*

Let $t_1 = (1, j_1, k_1, l_1)$. This node must have only one index in common with both c and d. Assume w.l.o.g. that $j_1 = 2$, $k_1 = 2$ and $l_1 = 3$. Let also $t_2 = (1, j_2, k_2, l_2)$ be the predecessor of c, i.e. $j_2, k_2, l_2 \neq 1$. Note that t_2 must be connected to node d and this is possible only if $l_2 = 2$. Node t_2 must also have one more index (other than i) in common with node t_1. Let w.l.o.g. $k_2 = 2$, i.e. $t_2 = (1, j_2, 2, 2)$.

Consider now the successor of d, namely $t_3 = (1, j_3, k_3, l_3)$, where $j_3, k_3 \neq 1$, $l_3 \neq 2$. Node t_3 can be connected to c (and not to d) only if $l_3 = 1$. Since $p \geq 3$, node t_3 must also be connected to nodes t_2 and t_1. This requires at least one of the following:

(i) $k_3 = 2$

(ii) $j_2 = j_3 = 2$

Consider Case (i), i.e. $t_3 = (1, j_3, 2, 1)$, Case (ii) to be treated in an analogous manner. Note that it might also be $j_2 = j_3$. The predecessor of t_2, namely $t_4 = (1, j_4, k_4, l_4)$ can be assumed not to coincide with the successor of t_3, since $p \geq 3$. This node must be connected to c, t_1, d and t_3. In order for node t_4 to be connected to c, at least one of its last three indices must be equal to 1. If $j_4 = 1$, then it must be $l_4 = 3$ for node t_4 to be connected to t_1. But then node t_4 cannot be connected to node t_3, since $k_3 \neq 2$.

If $k_4 = 1$, node t_4 must have index j in common with node t_3 and index l in common with node t_1 or vice versa. This implies $t_4 = (1,2,1,1)$ or $t_4 = (1, j_3, 1, 3)$. Finally, if $l_4 = 1$, $|t_1 \cap t_4| = 2$ implies $j_4 = 2$ and $|d \cap t_4| = 1$ implies $k_4 = 1$, i.e. it must be $t_4 = (1,2,1,1)$.

Overall, there are two possibilities for the form of node t_4, namely $t_4 = (1,2,1,1)$ or $t_4 = (1, j_3, 1, 3)$, $j_3 \neq 1$. We examine each of these two cases separately.

Case 2.34.2 *($t_4 = (1,2,1,1)$) Consider node $t_5 = (1, j_5, k_5, l_5)$.*

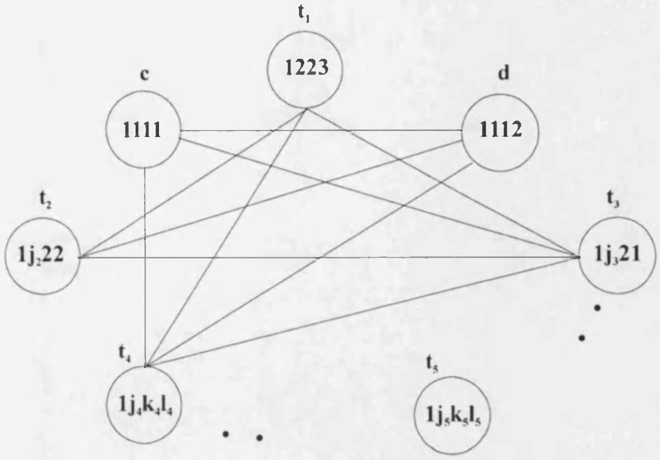


Figure 2.10: The antihole of Case 2.34.1

If $p = 3$, *this node must be the predecessor of* t_4 *and the successor of* t_3. *This implies that* t_5 *is connected to all nodes, except* t_3 *and* t_4. *Observe that* $|t_5 \cap c| = 2$ *and* $|t_5 \cap t_4| = 1$ *only if* $j_5 = 1$. *Since* $k_5 \neq 2$, *node* t_5 *can have only index* l *in common with* t_1, *i.e.* $l_5 = 3$. *The node* $t_5 = (1, 1, k_5, 3)$, *with* $k_5 \neq 1, 2$ *cannot be connected to nodes* t_2 *and* d.

If $p \geq 3$, *we can assume w.l.o.g. that node* t_5 *is neither the predecessor of* t_4 *nor the successor of* t_3, *i.e. it is connected to all nodes. Again, in order for* $|t_5 \cap c| \geq 2$, *at least one of* j_5, k_5, l_5 *must be equal to* 1. *It turns out that the only feasible option is* $t_5 = (1, 1, 2, 1)$. *In that case, there can exist no node consecutive to* t_5.

Case 2.34.3 *(*$t_4 = (1, j_3, 1, 3)$*) Consider node* $t_5 = (1, j_5, k_5, l_5)$.

If $p = 3$, *this node must again be the predecessor of* t_4 *and the successor of* t_3. *Observe that* $|t_5 \cap c| = 2$ *only if* $j_5 = 1$. *Node* $t_5 = (1, 1, k_5, l_5)$ *cannot be connected to node* t_1, *since* $k_5 \neq 2$ *and* $l_5 \neq 3$.

If $p \geq 3$, *we can again assume w.l.o.g. that node* t_5 *is connected to all nodes. In order for* $|t_5 \cap c| \geq 2$, *at least one of* j_5, k_5, l_5 *must be equal to* 1. *It is easy to see that all cases lead to infeasibility. For example, if* $j_5 = 1$, *and given that* $j_3 \neq 1$, *node* t_5 *will be connected to both* t_3 *and* t_4 *only if* $k_5 = l_5 = 1$, *which implies* $t_5 \equiv c$.

It follows that nodes c *and* d *cannot have three indices in common, if there is exactly one node between them.*

Case 2.34.4 *There is more than one node between nodes* c *and* d.

Consider first the predecessor and the successor of c, *namely* $c_1 = (1, j_1, k_1, l_1)$ *and* $c_2 = (1, j_2, k_2, l_2)$. *Since both nodes must be connected to* d *but not to* c, *it must be* $l_1 = l_2 = 2$. *The two nodes must have exactly two indices in common because of Case 2.34.1. Therefore* $j_1 \neq j_2$ *and* $k_1 \neq k_2$.

Secondly, let d_1 *and* d_2 *be the successor and the predecessor, respectively, of* d. *Following the same reasoning as above, both nodes must have index* l *equal to* 1. *Since* $p \geq 3$ *(i.e. there are at least seven nodes in* $\overline{H}$*), at least one of* d_1, d_2 *is not a neighbor of* c_1, c_2. *Assume w.l.o.g. the existence of at least one node between* c_1 *and* d_1. *This implies that* d_1 *must be connected to both* c_1 *and* c_2, *therefore let* $d_1 = (1, j_1, k_2, 1)$.

Let us examine two cases with respect to whether nodes c_2 *and* d_2 *are consecutive.*

***Case 2.34.5** (c_2 **and** d_2 **are consecutive**)* *In order for d_2 not be connected to c_2 and not to have three indices in common with d_1, d_2 must have index j different from j_1 and j_2. In addition, node d_2 must have index $k = k_1$, in order to be connected to node c_1. Let w.l.o.g. $d_1 = (1, j_3, k_1, 1)$. Figure 2.11 depicts this extended sequence of nodes.*

If $p = 3$, there must exist exactly one node between c_1 and d_1. This seventh node of the antihole, namely $t = (1, j_4, k_4, l_4)$, must be connected to all nodes except for nodes c_1 and d_1. This implies $k_4 \neq k_1, k_2$ and $l_4 \neq 1, 2$. It is easy to see that node t cannot be connected to both c_2 and d_2, since $j_2 \neq j_3$.

For $p > 3$, assume w.l.o.g. that node t is not a neighbor of c_1 or d_1. This implies that node t must be connected to all six nodes, which can be easily seen to be impossible.

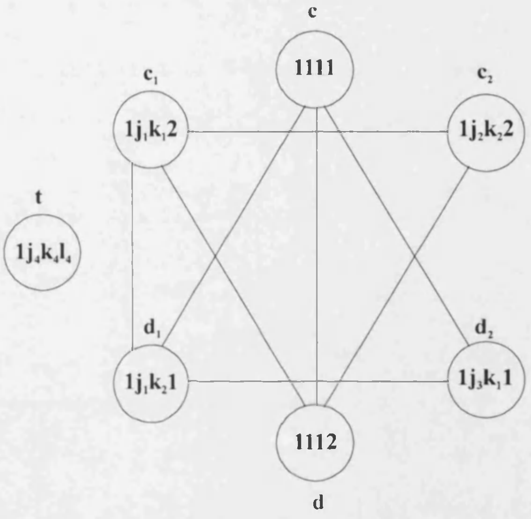


Figure 2.11: The antihole of Case 2.34.5

***Case 2.34.6** (c_2 **and** d_2 **are not consecutive**)* *Note that this case is possible only for $p > 3$. This case is illustrated at Figure 2.12. Assume again a node $t = (1, j_4, k_4, l_4)$. This node can be a neighbor of both c_1 and d_1 or only one of them or none of the men.*

Let t be a neighbor of neither c_1 nor d_1. For t to be connected to node c, at least one of j_4, k_4, l_4 must be equal to 1. If $j_4 = 1$, then it must be $k_4 = k_1$ or $l_4 = 2$ for $|t \cap c_1| = 2$. If $k_4 = k_1$, node $t = (1, 1, k_1, l_4)$ cannot be simultaneously connected to nodes c_2 and d_1. If $j_4 = 2$, node $t = (1, 1, k_4, 2)$ cannot be connected to both d_1 and d_2. The outcome of setting $k_4 = 1$ or $l_4 = 1$ is analogous, i.e. it cannot be that t is not a neighbor of none of c_1, d_1.

Let t be the predecessor of c_1 (the case of t being the successor of d_1 is symmetric). Since $l_4 \neq 2$, $|t \cap d| = 2$ only if $j_4 = 1$ or $k_4 = 1$. If $j_4 = 1$, $|t \cap c_2| = 2$ only if $k_4 = k_2$. As a result, node $t = (1, 1, k_2, l_4)$ can be connected to node d_2 only if $l_4 = 1$. But then $|t \cap c| = 3$, which contradicts to Case 2.34.1. Similarly, if $k_4 = 1$, $|t \cap c_2| = 2$ only if $j_4 = j_2$ and $|t \cap d_1| = 2$ only if $l_4 = 1$. Node $t = (1, j_2, 1, 1)$ has again three indices in common with node c, which again contradicts to Case 2.34.1.

Finally, assume that node t is a neighbor of both c_1 and d_1. Since $p > 3$, there are at least two nodes between c_2 and d_2. Consider the successor of c_2 and observe that this is a configuration symmetric to the case of t being a neighbor of c_1 only. It follows that this case is also infeasible.

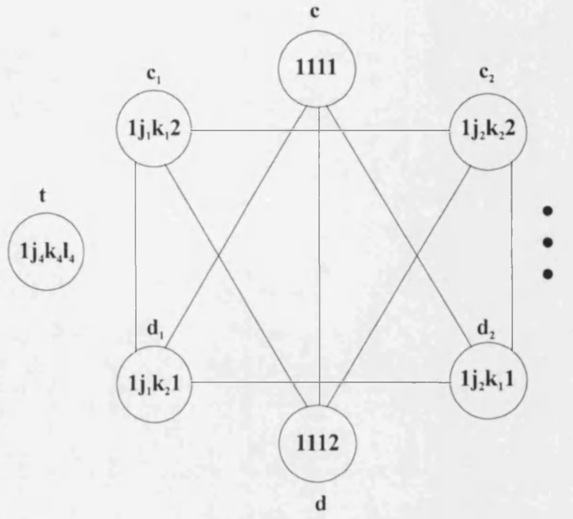


Figure 2.12: The antihole of Case 2.34.6

It follows that no pair of nodes of $\overline{H}$ can have three indices in common. ■

Proposition 2.35 *There exist no odd antiholes in G_A.*

Proof. Let $c = (1,1,1,1)$ and $d = (1,2,2,2)$ be two consecutive nodes of $\overline{H}$ and assume c_1 and d_1 to be the predecessor of c and the successor of d, respectively. Any node, except for c_1 and d_1, must have exactly two indices in common with both c and d. Observe that there exist only six possible forms of such nodes, namely $\{(1,1,2,l_1), (1,2,1,l_2), (1,1,k_1,2), (1,2,k_2,1), (1,j_1,1,2), (1,j_2,2,1)\}$, $j_i, k_i, l_i \notin \{1,2\}$ for $i = 1,2$. According to Prop. 2.34, at most one node can follow each one of these six formats. This places an upper bound of 10 on $|\overline{H}|$. Taking into account that $|\overline{H}|$ must also be odd, it follows that no antihole of size greater than 9 can exist.

If $|\overline{H}| = 9$, then exactly one node is selected from each of five formats. Assume that the five nodes are $\{(1,1,2,l_1), (1,2,1,l_2), (1,1,k_1,2), (1,2,k_2,1), (1,j_1,1,2)\}$. In a 9−antihole, each node is connected to 6 other nodes. Notice that node $(1,1,2,l_1)$ is connected to node $(1,1,k_1,2)$, to node $(1,2,1,l_2)$ only if $l_1 = l_2$, and to four other nodes (i.e. c, d, c_1, d_1) at most. Therefore, for a 9−antihole to exist, it must be $l_1 = l_2$. The same argument implies that $k_1 = k_2$.

The nodes are now $\{(1,1,2,l_1), (1,2,1,l_1), (1,1,k_1,2), (1,2,k_1,1), (1,j_1,1,2)\}$, $j_1, k_1, l_1 \notin \{1,2\}$. Consider again node $(1,1,2,l_1)$. This node cannot be the predecessor of c_1 or the successor of d_1, since this would imply that its degree is only 5. Hence, the only two options for the neighbors of node $(1,1,2,l_1)$ are nodes $(1,2,k_1,1)$ and $(1,j_1,1,2)$. Observe that these two nodes must be connected. This is possible only if $k_1 = 1$ or $j_1 = 2$, i.e. a contradiction. Therefore, no 9−antihole exists in G_A.

It remains to prove the non-existence of 7−antiholes. Such an antihole includes nodes c_1, c, d, d_1 and three more nodes, namely t_1, t_2 and t_3. Assume w.l.o.g. that $t_2 = (1,1,2,l_1)$. The options for t_1 and t_3 (i.e. the neighbors of t_2) are nodes $(1,2,1,l_2)$, $(1,2,k_2,1)$ and $(1,j_1,1,2)$. Since t_1 and t_3 must be connected, one of the nodes must definitely be $(1,2,1,l_2)$, because the other two nodes are not connected. This gives rise to two cases.

Let us examine first the case of $t_2 = (1,2,k_2,1)$ and $t_2 = (1,j_1,1,2)$, which is illustrated at Figure 2.13. Let $c_1 = (1,j_3,k_3,l_3)$. Since $j_3 \neq 2$, $|c_1 \cap d| = 2$ only if $k_3 = 2$ or $l_3 = 2$. If $l_3 = 2$, $|c_1 \cap t_2| = 2$ only if $j_3 = 1$ or $k_3 = 2$. The first option creates an edge between c_1 and c, whereas the second allows for nodes c_1 and d to have three indices in common, i.e. a contradiction.

If $k_3 = 2$, $|c_1 \cap t_3| = 2$ only if $j_3 = j_1$. This implies $c_1 = (1, j_1, 2, l_3)$, where $l_3 \notin \{1,2\}$. Consider the last node, i.e. $d_2 = (1, j_4, k_4, l_4)$. $|d_1 \cap c| = 2$ only if $j_4 = 1$ or $l_4 = 1$. If $j_4 = 1$ and given that $k_4 \notin \{1,2\}$, node d_1 cannot be connected to both c_1 and t_1. If $l_4 = 1$, $|d_1 \cap t_1| = 2$ only if $j_4 = 1$, which implies $|d_1 \cap c| = 3$, i.e. a contradiction to Prop. 2.33.

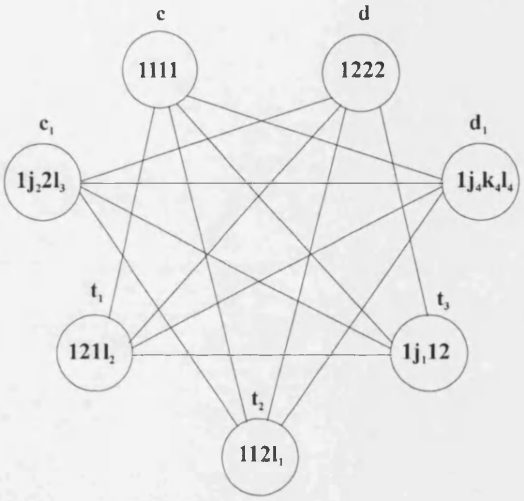


Figure 2.13: The antihole of Prop. 2.35

This result can be easily achieved for the remaining case. The proof is complete. ■

2.9 Concluding remarks

This chapter identified several classes of strong valid inequalities, all arising from the intersection graph of the OLS problem. This has been accomplished by exploiting problem-specific features, which arise from the inherent assignment structure of the problem. Essentially, all properties follow from the basic characteristic that two nodes are connected if and only if they have two or three indices in common. The subgraphs described include all cliques, a family of antiwebs, all wheels having double sets on their spokes, a family of composite cliques and two families of odd-holes, arising from odd-holes of the $3AAP_n$ and $3PAP_n$ problems, respectively. Most important, our analysis refers also to the form of lifted inequalities, which are maximal valid inequalities for the OLS polytope. The reader can find analogous results for related assignment problems in [9, 37]. Hence, the methodology presented here is possibly applicable to problems exhibiting or embedding an assignment structure. In Chapter 7, we apply similar ideas in order to obtain a class of cliques for the multi-index assignment problem.

Chapter 3

Polyhedral characterisation of the OLS polytope

3.1 Introduction and preliminaries

This chapter presents a polyhedral characterisation of the OLS problem. The dimension of the associated polytope P_I is established and families of valid inequalities presented in the previous chapter are further analysed from a polyhedral perspective. This amounts to examining which of these inequalities are facets of P_I, identifying their Chvátal rank and providing separation algorithms of low complexity.

We briefly summarize some basic concepts and definitions of polyhedral theory ([47, 74, 68]). A *polyhedron* is the intersection of a finite number of half spaces. A *polytope* is a bounded polyhedron. A polytope P is of *dimension* n, denoted as $\dim(P) = n$, if it contains $n + 1$ affinely independent points. By convention, if $P = \emptyset$ then $\dim(P) = -1$. If $P = \{x \in \mathbb{R}^n : B^{=} x = b^{=}, B^{\leq} x \leq b^{\leq}\}$ then

$$dim(P) = n - rank(B^{=}) \tag{3.1}$$

where $B^{=}$ and $B^{\leq}$ denote the matrix of coefficients of equality and "less-than-or-equal-to" type inequality constraints respectively, while $b^{=}$ and $b^{\leq}$ denote the corresponding right-hand side vectors for the linear system defining P. An inequality $ax \leq a_0$ is called *valid* for P, if it is satisfied by all $x \in P$. It is called *supporting,* if it is valid and there exist $\hat{x} \in P$ satisfying $a\hat{x} = a_0$. The set of points, which satisfy $ax \leq a_0$ as equality ($F = \{x \in P : ax = a_0\}$), is called a *face* of P. A face F of a polytope P is said to be *improper,* if $ax = a_0$ for all $x \in P$. A proper, non-empty face F of P is called a *facet,* if $\dim(F) = \dim(P) - 1$. Facets are important, since they provide a minimal inequality representation of a polyhedron. Our main interest in this chapter is in the facets of the convex hull P_I of integer points in P_L, defined in Section 2.1. Conditions *(c)* and *(d)* of the following theorem usually provide the two basic tools for proving that a given inequality $ax \leq a_0$ defines a facet.

Theorem 3.1 *(see [74, Theorem 3.16])Let $P \subseteq \mathbb{R}^n$ be a polyhedron and assume that B is a real valued $m \times n$ matrix and $b \in \mathbb{R}^m$ such that $P = \{x \in \mathbb{R}^n : B^{=} x = b^{=}, B^{\leq} x \leq b^{\leq}\}$ where $B = (B^{=}, B^{\leq})^T$ and $b = (b^{=}, b^{\leq})^T$. Let F be a non-empty face of P, then the following statements are equivalent:*

(a) F is a facet of P.

(b) F is a maximal proper face of P.

(c) $dim(F) = dim(P) - 1$.

(d) There exists an inequality $ax \leq a_0$ valid with respect to P with the following three properties:

(i) $F = \{x \in P : ax = a_0\}$.

(ii) There exists $\bar{x} \in P$ with $a\bar{x} < a_0$, i.e. the inequality is proper

If any other inequality $dx \leq d_0$, valid with respect to P satisfies $F = \{x \in P : dx = d_0\}$, then (d, d_0) can be expressed as a linear (affine) combination of $(B^=, b^=)$ and (a, a_0).

Condition *(d)* has been used in [2] to prove that cliques of type II and III (Section 2.3) induce facets of P_I. In this chapter, we apply the same approach to prove that composite clique inequalities also define facets of P_I (Section 2.6). We also apply condition *(d)* to prove that odd cycles of type II (Section 2.7.2) induce facets of the Latin square polytope or $3PAP_n$ ([37]). The dimension of P_I is also established in a similar manner. A similar approach has been used for establishing the dimension and facet-defining inequalities of the $3AAP_n$, and $3PAP_n$ polytopes in [9] and [37], respectively (Section 2.1).

Recall the IP model for the OLS problem and the definitions of polytopes $P_I, \tilde{P}_I$ and P_L from Section 2.1. The following remark will be used for establishing the dimension of P_L and P_I.

Remark 3.1 *Constraints (2.1), (2.2), (2.3), for a given value of the l index, are equivalent to the constraints of a $3AAP_n$.*

Let us first discuss some properties of $\tilde{P}_I$. Since the *OLS* packing polytope $\tilde{P}_I$ is a special case of P_{SP}, $\tilde{P}_I$ shares the following properties of P_{SP} ([18]):

(i) $\tilde{P}_I$ is full-dimensional, i.e. $dim(\tilde{P}_I) = n^4$. The $n^4 + 1$ independent points of $\tilde{P}_I$ are the zero vector and all the n^4 unit vectors.

(ii) $\tilde{P}_I$ is down monotone, i.e. $x \in \tilde{P}_I \Rightarrow y \in \tilde{P}_I$ for all y such that $0 \leq y \leq x$.

(iii) The non-negativity constraints $x_{ijkl} \geq 0$ define facets of $\tilde{P}_I$.

Although we know quite a few things about the facial structure of $\tilde{P}_I$, the same cannot be said with respect to P_I. Since P_I is a face of P, we know that $dim(P_I) \leq dim(\tilde{P}_I)$. However, the structure of P_I presents irregularities that do not appear in $\tilde{P}_I$. For example, we know that $P_I = \emptyset$ for $n = 2$ and $n = 6$. $P_I^2 = \emptyset$ can be easily verified, since there are only two Latin squares for $n = 2$. As stated previously, $P_I^6 = \emptyset$ was proven in [81]. Fortunately, $P_I^n \neq \emptyset$ for $n \neq 2, 6$ as shown in [20] (see also [60, Theorem 2.9]).

Moreover, a complete characterisation of all the facets defining a polytope arising from a $0-1$ IP is possible only if $\mathcal{NP} = Co\mathcal{NP}$, as analysed in [68, p.139-141]. Hence, the complexity of the problem restricts any attempt to achieve a complete description of the convex hull of integer points. A partial polyhedral description of the OLS polytope appears to be the only reasonable aim. Even such a result, however, can be sufficient in order to solve the problem with a cutting plane algorithm incorporated within a *Branch & Cut* algorithmic framework.

The remainder of this chapter proceeds as follows. Section 3.2 identifies the dimension of both P_L and P_I. Section 3.3 examines which of the constraints defining P_L define facets of P_I. The following sections describe the inequalities induced by sub-graphs of G_A, examine their Chvátal rank ([25]) and provide separation algorithms. Section 3.7.2 introduces a new class of facet-defining inequalities for the polytope of $3PAP_n$.

3.2 The dimension of P_I

A necessary step towards achieving a polyhedral description of polytope P_I is to establish its dimension. Before establishing the dimension of P_I, the dimension of P_L is first provided. According to eq. (3.1), it suffices to count the rank of the minimum equality system of P_L. Throughout the remainder of this chapter, we assume that each of the four sets I, J, K, L is equal to $\{1, ..., n\}$ or $\{0, ..., n-1\}$. It is easy to see that this has been done only for convenience of the presentation.

Theorem 3.2 *The rank of the system* $Ax = e$ *is* $6 \cdot (n-1)^2 + 4 \cdot (n-1) + 1$

Proof. Order the n^4 columns of the A matrix, denoted by x_{ijkl}, so that index i is the slowest to vary and index l is the quickest. For $n = 2$, the order of the column indices is:

$(1,1,1,1),(1,1,1,2),(1,1,2,1),(1,1,2,2),(1,2,1,1),(1,2,1,2),(1,2,2,1),(1,2,2,2),$
$(2,1,1,1),(2,1,1,2),(2,1,2,1),(2,1,2,2),(2,2,1,1),(2,2,1,2),(2,2,2,1),(2,2,2,2)$

As to the $6n^2$ rows, we divide them into six sets of n^2 rows each, as defined by equalities (2.1)..(2.6).

Figure 3.1 illustrates the matrix for $n = 2$. Note that, for $n = 2$, each of the 6 constraint sets has 4 equality constraints, each involving 4 variables.

1	1				1				1				1				(k,l)=(1,1)
2		1				1				1				1			(k,l)=(1,2)
3			1				1				1				1		(k,l)=(2,1)
4				1				1				1				1	(k,l)=(2,2)
5	1		1						1		1						(j,l)=(1,1)
6		1		1						1		1					(j,l)=(1,2)
7					1		1						1		1		(j,l)=(2,1)
8						1		1						1		1	(j,l)=(2,2)
9	1	1							1	1							(j,k)=(1,1)
10			1	1							1	1					(j,k)=(1,2)
11					1	1							1	1			(j,k)=(2,1)
12							1	1							1	1	(j,k)=(2,2)
13	1		1		1		1										(i,l)=(1,1)
14		1		1		1		1									(i,l)=(1,2)
15									1		1		1		1		(i,l)=(2,1)
16										1		1		1		1	(i,l)=(2,2)
17	1	1			1	1											(i,k)=(1,1)
18			1	1			1	1									(i,k)=(1,2)
19									1	1			1	1			(i,k)=(2,1)
20											1	1			1	1	(i,k)=(2,2)
21	1	1	1	1													(i,j)=(1,1)
22					1	1	1	1									(i,j)=(1,2)
23									1	1	1	1					(i,j)=(2,1)
24													1	1	1	1	(i,j)=(2,2)

Figure 3.1: The matrix A for $n = 2$

Row set	*Rows removed*	*Rows removed for n=2*
(2.1)		
(2.2)	n^2+1	5
(2.3)	$2n^2+1$	9
(2.4)	$3n^2+1$	13
(2.5)	$4n^2+1$	17
(2.6)	$5n^2+1$	21

Table 3.1: Linearly dependent rows removed at Step I

Row set	*Rows removed*	*Rows removed for n=2*
(2.1)		
(2.2)	n^2+1	5
(2.3)	$2n^2+1$	9
(2.4)	$3n^2+1$	13
(2.5)	$\{4n^2+(t-1)\cdot n+1,\, t=2..n\}, 4n^2+1$	$17, 19$
(2.6)	$\{5n^2+(t-1)\cdot n+1,\, t=2..n\}, 5n^2+1$	$21, 23$

Table 3.2: Linearly dependent rows removed at Steps I - II

To find the rank of the A matrix, we follow five steps, the last four of which identify a $3AAP_n$ substructure, exactly as at Remark 3.1.

Step I: It is obvious that the sum of all the rows of each set is the same, i.e. $\sum\{x_{ijkl} : i \in I, j \in J, k \in K, l \in L\} = n^2$. Therefore, any one constraint can be removed from any of the six sets as being linearly dependent. We choose to keep the row set (2.1) intact and remove the first row of all the remaining sets. Table 3.1 illustrates the outcome.

Step II: Consider row sets (2.4), (2.5) and (2.6). Observe that, as noted in Remark 3.1, they form n independent $3AAP_n$ problems, one for each value of the index i. For $i = i_0$, the corresponding $3AAP_n$ involves the n^3 variables x_{i_0jkl}, for $j, k, l = 1, ..., n$, and the $3n$ rows $3n^2 + (i_0 - 1) \cdot n + t$, $4n^2 + (i_0 - 1) \cdot n + t$, $5n^2 + (i_0 - 1) \cdot n + t$, for $t = 1, ..., n$. It is shown in [9] that the rank of a $3n \times n^3$ $3AAP_n$ matrix is $3n - 2$. Hence, we can remove up to 2 rows from each of the n $3AAP_n$ problems. Note that, having removed rows $n^2 + 1$, $2n^2 + 1$ at Step I, the $3AAP_n$ corresponding to $i_0 = 1$ includes no linearly dependent rows. For the remaining $n - 1$ independent $3AAP_n$ problems, we can remove two linearly dependent rows. We choose to remove rows numbered $4n^2+(t-1)\cdot n+1$, $5n^2 + (t - 1) \cdot n + 1$, for $t = 2, ..., n$, i.e. a total of $2(n - 1)$ rows. Table 3.2 gives a complete list of rows removed up to this point.

Step III: Consider row sets (2.2), (2.3) and (2.6). Observe that they form n independent $3AAP_n$ problems, one for each value of the index j. For $j = j_0$, the corresponding $3AAP_n$ involves variables x_{ij_0kl}, for $i, k, l = 1...n$, and rows $n^2+(j_0-1)\cdot n+t+j_0$, $2n^2+(j_0-1)\cdot n+t$, $5n^2+(t-1)\cdot n+j_0$, for $t = 1, ..., n$. Again, for $j_0 = 1$, the corresponding $3AAP_n$ includes no linearly dependent rows, since rows $2n^2 + 1$, $5n^2 + 1$ have already been removed. All the rows of the remaining $n - 1$ independent $3AAP_n$ problems are present. We choose to remove two linearly dependent rows from each problem, namely rows $2n^2 + (t - 1) \cdot n + 1$, $5n^2 + t$, for $t = 2, ..., n$, i.e. $2(n - 1)$ rows in total. Table 3.3 gives a complete list of rows removed.

Step IV: Consider row sets (2.1), (2.3) and (2.5). Observe that they form n independent $3AAP_n$ problems, one for each value of the index k. For $k = k_0$, the corresponding $3AAP_n$ involves variables x_{ijk_0l}, for $i, j, l = 1, ..., n$ and rows $(k_0 - 1) \cdot n + t$, $2n^2 + (t - 1) \cdot n + k_0$, $4n^2 + (t - 1) \cdot n + k_0$, for $t = 1, ..., n$. For $k_0 = 1$, the corresponding $3AAP_n$ includes no linearly dependent rows, since rows

Row set	*Rows removed*	*Rows removed for n=2*
(2.1)		
(2.2)	n^2+1	5
(2.3)	$\{2n^2+(t-1)\cdot n+1, t=2..n\}, 2n^2+1$	9, 11
(2.4)	$3n^2+1$	13
(2.5)	$\{4n^2+(t-1)\cdot n+1, t=2..n\}, 4n^2+1$	17, 19
(2.6)	$\{5n^2+(t-1)\cdot n+1, t=2..n\}, \{5n^2+t, t=1..n\}$	21, 22, 23

Table 3.3: Linearly dependent rows removed at Steps I - III

Row set	*Rows removed*	*Rows removed for n=2*
(2.1)		
(2.2)	n^2+1	5
(2.3)	$\{2n^2+(t-1)\cdot n+1, t=2..n\}, \{2n^2+t, t=1..n\}$	9, 10, 11
(2.4)	$3n^2+1$	13
(2.5)	$\{4n^2+(t-1)\cdot n+1, t=2..n\}, \{4n^2+t, t=1..n\}$	17, 18, 19
(2.6)	$\{5n^2+(t-1)\cdot n+1, t=2..n\}, \{5n^2+t, t=1..n\}$	21, 22, 23

Table 3.4: Linearly dependent rows removed at Steps I - IV

$2n^2+1$, $4n^2+1$ have already been removed. The remaining $n-1$ independent $3AAP_n$ problems have been left intact. We choose to remove two linearly dependent rows from each problem, namely rows $2n^2+t$, $4n^2+(t-1)\cdot n$, for $t=2...n$, i.e. $2(n-1)$ rows. Table 3.4 gives a complete list of rows removed so far.

Step V: Consider row sets (2.1), (2.2) and (2.4). Observe that they form n independent $3AAP_n$ problems one for each value of the index l. For $l=l_0$, the corresponding $3AAP_n$ involves variables x_{ijkl_0}, for $i,j,k=1,...,n$, and rows $(t-1)\cdot n+l_0$, $n^2+(t-1)\cdot n+l_0$, $3n^2+(t-1)\cdot n+l_0$, for $t=1,...,n$. Again for $l_0=1$, the corresponding $3AAP_n$ includes no linearly dependent rows, since rows n^2+1, $3n^2+1$ have already been removed. All the rows of the remaining $n-1$ independent $3AAP_n$ problems are present. We choose to remove two linearly dependent rows from each problem, namely rows n^2+t, $3n^2+t$, for $t=2,...,n$ i.e. $2(n-1)$ rows. Table 3.5 gives a complete list of rows removed so far.

In total, $4\cdot 2\cdot(n-1)+5=8n-3$ rows have been removed. Therefore, $6n^2-8n+3$ is an upper bound on the rank of A. We will complete the proof by exhibiting $6n^2-8n+3$ affinely independent columns.

Consider the columns:

Row set	*Rows removed*	*Rows removed for n=2*
(2.1)		
(2.2)	$\{n^2+t, t=1..n\}$	5, 6
(2.3)	$\{2n^2+(t-1)\cdot n+1, t=2..n\}, \{2n^2+t, t=1..n\}$	9, 10, 11
(2.4)	$\{3n^2+t, t=1..n\}$	13, 14
(2.5)	$\{4n^2+(t-1)\cdot n+1, t=2..n\}, \{4n^2+t, t=1..n\}$	17, 18, 19
(2.6)	$\{5n^2+(t-1)\cdot n+1, t=2..n\}, \{5n^2+t, t=1..n\}$	21, 22, 23

Table 3.5: Linearly dependent rows removed at Steps I - V

$(1,1,1,1),...,(1,1,n,1),...,(1,1,1,n),....,(1,1,n,n)$ (n^2columns)
$(2,1,1,1),...,(n,1,1,1),...,(2,1,1,n),....,(n,1,1,n)$ ($n(n-1)$ columns)
$(1,2,1,1),...,(1,n,1,1),...,(1,2,1,n),....,(1,n,1,n)$ ($n(n-1)$ columns)
$(2,2,1,1),...,(n,n,1,1),...,(2,2,1,n-1),....,(n,n,1,n-1)$ ($(n-1)^2$ columns)
$(1,2,2,1),...,(1,n,n,1),...,(1,2,2,n-1),....,(1,n,n,n-1)$ ($(n-1)^2$ columns)
$(2,1,2,1),...,(n,1,n,1),...,(2,1,2,n-1),....,(n,1,n,n-1)$ ($(n-1)^2$ columns)

The matrix formed by these columns and the $6n^2 - 8n + 3$ remaining rows of A is square upper triangular, with each diagonal element equal to 1, therefore non-singular. This matrix is illustrated, for $n = 2$, in Figure 3.2.

1	1				1			1			1	(k,l)=(1,1)
2		1				1			1			(k,l)=(1,2)
3			1				1			1		(k,l)=(2,1)
4				1								(k,l)=(2,2)
7					1		1				1	(j,l)=(2,1)
8						1						(j,l)=(2,2)
12							1					(j,k)=(2,2)
15								1		1	1	(i,l)=(2,1)
16									1			(i,l)=(2,2)
20										1		(i,k)=(2,2)
24											1	(i,j)=(2,2)

Figure 3.2: The upper triangular matrix for $n = 2$

It is easy to verify that $6n^2 - 8n + 3 = 6 \cdot (n-1)^2 + 4 \cdot (n-1) + 1$. ■

Corollary 3.3 $dim(P_L) = (n-1)^4 + 4 \cdot (n-1)^3$.

Proof. By Newton's polynomial $n^4 = [(n-1)+1]^4 = (n-1)^4 + 4 \cdot (n-1)^3 + 6 \cdot (n-1)^2 + 4 \cdot (n-1) + 1$. The result follows from eq. 3.1. ■

We describe the tools needed to obtain the dimension of P_I.

Unless otherwise stated, we illustrate a pair of *OLS* as points of P_I expressed in terms of four sets of indices, viz. I for the row set, J for the column set, and K and L for the set of elements of the first and the second Latin square respectively. The elements of all four sets are the integers from 1 to n. We will further use $k(i,j)$ (respectively $l(i,j)$) to denote the value of the cell in row i, column j of the first (second) Latin square. Thus $k(i,j) \in K$ and $l(i,j) \in L$. The following remark reveals a very useful property of the points of P_I corresponding to a pair of *OLS.*

Remark 3.2 *Given an* OLS *structure and* $m_1, m_2 \in M$*, where* M *can be any one of the disjoint* $n-$ *sets* I, J, K, L*, (inter)changing all* m_1 *values to* m_2 *and all* m_2 *values to* m_1 *yields another* OLS *structure. These two structures are called* equivalent *([34, p. 168]).*

Remark 3.3 *Given an* OLS *structure and two sets* M_1, M_2*, where* M_1, M_2 *can be any of the four disjoint* $n-$ *sets* I, J, K, L*, (inter)changing the roles of sets* M_1, M_2 *yields another* OLS *structure. These two pairs of OLS are called* conjugate.

Remark 3.3 follows from the definition of the conjugate(s) of a Latin square, as described also in [34, p. 168]. As an example, if we interchange the roles of the row and column sets in a Latin square, the resulting square will be the transpose of the first. In exactly the same way, interchanging the roles of the row and column in a pair of OLS, the result is two squares, which are the transposes of the initial ones. It is easy to see that these squares remain orthogonal.

If the interchange is carried out for elements of set I (i.e. $M = I$) we call it a first index interchange, for elements of set J a second index interchange, etc. To facilitate the application of interchanges, we define the *interchange* operator ($\leftrightarrow$). By setting $x^* = x(i_1 \leftrightarrow i_2)_1$, we imply that at point $x \in P_I$ we apply a first index interchange between rows $i_1, i_2 \in I$, deriving point $x^* \in P_I$. Note that brackets must also have an index for denoting the set of the indices that are interchanged. Notation without this subscript in cases like $(1 \leftrightarrow n)$ becomes ambiguous. It is easy to see that $x(m_1 \leftrightarrow m_2)_t = x(m_2 \leftrightarrow m_1)_t$, i.e. the operator is associative, and $x = x(m_1 \leftrightarrow m_2)_t(m_1 \leftrightarrow m_2)_t$. A series of interchanges at a point $x \in P$ is expressed by using the operator ($\leftrightarrow$) as many times as the number of interchanges with priority from left to right. For example, $x^* = x(i_1 \leftrightarrow i_2)_1(1 \leftrightarrow n)_3$ is taken to mean that at point x we apply a first index interchange between $i_1, i_2 \in I$ and then at the derived point we apply a third index interchange between $1, n \in K$, thus yielding point x^*. It is clear that the interchange operator is also commutative, i.e. the order in which the interchanges are applied is not important. The properties of the interchange operator are listed below.

Property I: $x = x(m_1 \leftrightarrow m_2)_t(m_1 \leftrightarrow m_2)_t$

Property II: $x(m_1 \leftrightarrow m_2)_t = x(m_2 \leftrightarrow m_1)_t$ (associativity)

Property III: $x(m_1 \leftrightarrow m_2)_t(m_2 \leftrightarrow m_3)_t = x(m_2 \leftrightarrow m_3)_t(m_1 \leftrightarrow m_2)_t$ (commutativity)

We additionally define a *conditional* interchange as an interchange to be performed only when a certain condition is met. The condition refers to a logical expression consisting of values of an index set, at a given point x. If this expression evaluates *true* then the interchange will be applied to point x. Since we are going to use only conditional interchanges for which both the logical expression and the interchange refer to elements of the same set we will use a common subscript for both. Thus to denote this type of interchange at point x we will use the expression $x(\mathit{condition?interchange})_t$, where the subscript refers to the index set. For example, $x_2 = x_1(i_1 = n?1 \leftrightarrow i_1)_1$ implies that if $i_1 = n$ at point x_1 then we derive point x_2 by applying at point x_1 the first index interchange between $1, i_1$. If $i_1 \neq n$ then $x_2 = x_1$. As in the previous case we can have more than one conditional interchange in the same expression. Again priority is considered from left to right. As we will see shortly, the interchanges are extensively used for establishing the dimension of P_I and for proving that certain inequalities are facet defining.

The interchange operator is also used to illustrate the derivation of a conjugate point. Given a point $x \in P_I$, writing $x^* = x(I \longleftrightarrow K)$ implies that point x^* is derived by interchanging the roles of sets I and K at point x. The notion of a conditional interchange is also applicable in the case of conjugate points.

An additional complication for proving the dimension of P_I comes from the fact that it is not easy to exhibit a pair of *OLS* for every value of n, i.e. for every n it is difficult to demonstrate a $0-1$ vector feasible with respect to constraints (2.1),...,(2.6) that will have specific variables set to one. In contrast, both for $3AAP_n$ and $3PAP_n$ such "trivial" points exist. For the $3AAP_n$ the trivial point has $x_{ijk} = 1$ for $i = j = k$ and $x_{ijk} = 0$ for $i \neq j \neq k \neq i$. For the $3PAP_n$ the trivial solution is the Latin square having x_{ijk} equal to 1 if $k = i + j - 1 \bmod n$ and equal to 0 otherwise. To overcome this difficulty, in the following lemma we establish, for $n \geq 4$ and $n \neq 6$, the existence of an *OLS* structure with four specific variables set to one.

Table 3.6: Point x_0

	1	$\cdots$	j_0	$\cdots$
1	1		k_1	
$\vdots$				
i_0	k_2		k_3	
$\vdots$				

$\cdots$

	1	$\cdots$	j_0	$\cdots$
1	1		l_1	
$\vdots$				
i_0	l_2		l_3	
$\vdots$				

Table 3.7: An arbitrary point $x \in P_I$ (Lemma 3.4)

	1	$\cdots$	j_0	$\cdots$
1	k_b		k_c	
$\vdots$				
i_0	k_d		k_e	
$\vdots$				

$\cdots$

	1	$\cdots$	j_0	$\cdots$
1	l_b		l_c	
$\vdots$				
i_0	l_d		l_e	
$\vdots$				

Lemma 3.4 *For $n \geq 4$ and $n \neq 6$ let $i_0 \in I \setminus \{1\}$, $j_0 \in J \setminus \{1\}$, $k_1, k_2, k_3 \in K \setminus \{1\}$, $k_3 \neq k_1, k_2$ and $l_1, l_2, l_3 \in L \setminus \{1\}$, $l_3 \neq l_1, l_2$. Then there exists a point $x_0 \in P_I$ with four particular variables taking value one as illustrated in Table 3.6.*

Proof. Consider an arbitrary point $x \in P_I$ as illustrated in Table 3.7.

where $k_b, k_c, k_d, k_e \in K$, $l_b, l_c, l_d, l_e \in L$. For x to be a valid *OLS* structure we must have $k_b \neq k_c, k_d$, $k_e \neq k_c, k_d$, $l_b \neq l_c, l_d$, $l_e \neq l_c, l_d$. It is easy to see that $x_0 = x(k_b \neq 1?k_b \leftrightarrow 1)_3(l_b \neq 1?l_b \leftrightarrow 1)_4$, if we additionally set $k_c = k_1$, $k_d = k_2$, $k_e = k_3$ and $l_c = l_1$, $l_d = l_2$, $l_e = l_3$. ■

There might be more than one points of P_I with these four variables set to one. However, one point suffices to carry out the following proofs.

Since $P_I \subset P_L$, $\dim(P_I) \leq \dim(P_L)$. Moreover, $\dim(P_I) < \dim(P_L)$ if and only if there exists an equation $ax = a_0$ satisfied by all $x \in P_I$ such that it is not implied by (i.e. cannot be expressed as a linear combination of) the equations $Ax = e$. We will show that no such equation exists.

Theorem 3.5 *Let $n \geq 4$ and $n \neq 6$, and suppose every $x \in P_I^n$ satisfies $ax = a_0$ for some $a \in \mathbb{R}^{n^4}$, $a_0 \in \mathbb{R}$. Then there exist $6n^2$ scalars λ^1_{kl}, λ^2_{il}, λ^3_{jl}, λ^4_{ij}, λ^5_{jk}, λ^6_{ik}, $i \in I$, $j \in J$, $k \in K$, $l \in L$, satisfying*

$$a_{ijkl} = \lambda^1_{kl} + \lambda^2_{il} + \lambda^3_{jl} + \lambda^4_{ij} + \lambda^5_{jk} + \lambda^6_{ik} \tag{3.2}$$

$$\begin{aligned} a_0 &= \sum\{\lambda^1_{kl} : k \in K, l \in L\} + \sum\{\lambda^2_{il} : i \in I, l \in L\} \\ &+ \sum\{\lambda^3_{jl} : j \in J, l \in L\} + \sum\{\lambda^4_{ij} : i \in I, j \in J\} \\ &+ \sum\{\lambda^5_{jk} : j \in J, k \in K\} + \sum\{\lambda^6_{ik} : i \in I, k \in K\} \end{aligned} \tag{3.3}$$

Proof. Define
$\lambda^1_{kl} = a_{11kl}$
$\lambda^2_{jl} = a_{1j1l} - a_{111l}$
$\lambda^3_{jk} = a_{1jk1} - a_{1j11} - a_{11k1} + a_{1111}$
$\lambda^4_{il} = a_{i11l} - a_{111l}$
$\lambda^4_{ij} = a_{ij11} - a_{i111} - a_{1j11} + a_{1111}$
$\lambda^4_{ik} = a_{i1k1} - a_{i111} - a_{11k1} + a_{1111}$

Table 3.8: Point x Proposition 3.6)

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		b		c	
$\vdots$					
i_2		d		e	
$\vdots$					

$\cdots$

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		p		q	
$\vdots$					
i_2		r		s	
$\vdots$					

Note that these $6n^2$ scalars are defined in such a way that exactly $8n-3$ of them, corresponding to the linearly dependent rows of the A matrix, equal zero.

By substituting the λs in equation (3.2) we get:

$$\begin{aligned} a_{ijkl} &= a_{11kl} + a_{i11l} + a_{1j1l} + a_{ij11} + a_{1jk1} + a_{i1k1} \\ &\quad -2a_{i111} - 2a_{1j11} - 2a_{11k1} - 2a_{111l} + 3a_{1111} \end{aligned} \tag{3.4}$$

Substitution alone is enough to show that (3.4) is true for a_{1111} and for all cases where at least two of the indices are equal to one. For all cases, where only one of the indices equals one, equation (3.4) becomes

$$a_{ijk1} = a_{ij11} + a_{i1k1} + a_{1jk1} - a_{i111} - a_{1j11} - a_{11k1} + a_{1111} \tag{3.5}$$

$$a_{ij1l} = a_{ij11} + a_{i11l} + a_{1j1l} - a_{i111} - a_{1j11} - a_{111l} + a_{1111} \tag{3.6}$$

$$a_{i1kl} = a_{i1k1} + a_{i11l} + a_{11kl} - a_{i111} - a_{11k1} - a_{111l} + a_{1111} \tag{3.7}$$

$$a_{1jkl} = a_{1jk1} + a_{1j1l} + a_{11kl} - a_{1j11} - a_{11k1} - a_{111l} + a_{1111} \tag{3.8}$$

Before proving that eq.(3.4) to (3.8) hold, we prove the following proposition.

Proposition 3.6 *For $n \geq 3$ and $n \neq 6$, it can be shown that*

$$\begin{aligned} & a_{i_1j_1k(i_1,j_1)l(i_1,j_1)} + a_{i_1j_2k(i_1,j_2)l(i_1,j_2)} + a_{i_2j_1k(i_2,j_1)l(i_2,j_1)} + a_{i_2j_2k(i_2,j_2)l(i_2,j_2)} \\ & +a_{i_1j_1k(i_2,j_2)l(i_2,j_2)} + a_{i_1j_2k(i_2,j_1)l(i_2,j_1)} + a_{i_2j_1k(i_1,j_2)l(i_1,j_2)} + a_{i_2j_2k(i_1,j_1)l(i_1,j_1)} \\ & = a_{i_1j_1k(i_2,j_1)l(i_2,j_1)} + a_{i_1j_2k(i_2,j_2)l(i_2,j_2)} + a_{i_2j_1k(i_1,j_1)l(i_1,j_1)} + a_{i_2j_2k(i_1,j_2)l(i_1,j_2)} \\ & +a_{i_1j_1k(i_1,j_2)l(i_1,j_2)} + a_{i_1j_2k(i_1,j_1)l(i_1,j_1)} + a_{i_2j_1k(i_2,j_2)l(i_2,j_2)} + a_{i_2j_2k(i_2,j_1)l(i_2,j_1)} \end{aligned} \tag{3.9}$$

for $i_1, i_2 \in I$, $i_1 \neq i_2$ and $j_1, j_2 \in J$, $j_1 \neq j_2$.

Proof. Let $b, c, d, e \in K$ and $p, q, r, s \in L$ and consider an arbitrary point $x \in P_I$ as illustrated in Table 3.8.

Let $x' = x(i_1 \leftrightarrow i_2)_1$ (Table 3.9).

Let $\bar{x} = x(j_1 \leftrightarrow j_2)_2$ (Table 3.10).

Let $\bar{x}' = \bar{x}(i_1 \leftrightarrow i_2)_1$ (Table 3.11).

Essentially, we interchange: rows i_1 and i_2 at point x to derive point x'; columns j_1 and j_2 at point x to derive point $\bar{x}$; finally, rows i_1 and i_2 but this time at point $\bar{x}$ to derive point $\bar{x}'$. These simple interchanges allow for completing this proof quite directly and greatly simplify the following proofs.

Table 3.9: Point x' (Proposition 3.6)

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		d		e	
$\vdots$					
i_2		b		c	
$\vdots$					

$\cdots$

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		r		s	
$\vdots$					
i_2		p		q	
$\vdots$					

Table 3.10: Point $\overline{x}$ (Proposition 3.6)

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		c		b	
$\vdots$					
i_2		e		d	
$\vdots$					

$\cdots$

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		q		p	
$\vdots$					
i_2		s		r	
$\vdots$					

Table 3.11: Point $\overline{x}'$ Proposition 3.6)

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		e		d	
$\vdots$					
i_2		c		b	
$\vdots$					

$\cdots$

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$
$\vdots$					
i_1		s		r	
$\vdots$					
i_2		q		p	
$\vdots$					

Let $k^x(i_s, j_t)$ $(l^x(i_s, j_t))$ denote the value of the $k(l)$ index for $i = i_s, : j = j_t$ at point x. $k^{x'}(i_s, j_t)$, $k^{\bar{x}}(i_s, j_t)$, $k^{\bar{x}'}(i_s, j_t)$ and $l^{x'}(i_s, j_t)$, $l^{\bar{x}}(i_s, j_t)$, $l^{\bar{x}'}(i_s, j_t)$ are defined accordingly for points x', $\bar{x}$ and $\bar{x}'$.

Since $x, x' \in P_I$ we have $ax = ax'$. By observing that all a_{ijkl} terms for every $i \in I \setminus \{i_1, i_2\}$ are canceled out (all rows except for i_1, i_2 have been left intact), $ax = ax'$ becomes

$$\begin{array}{rcl}
a_{i_1 j_1 bp} + a_{i_1 j_2 cq} + \sum\{a_{i_1 j k^x(i_1,j) l^x(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 dr} + a_{i_2 j_2 es} + \sum\{a_{i_2 j k^x(i_2,j) l^x(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\} = \\
a_{i_1 j_1 dr} + a_{i_1 j_2 es} + \sum\{a_{i_1 j k^{x'}(i_1,j) l^{x'}(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 bp} + a_{i_2 j_2 cq} + \sum\{a_{i_2 j k^{x'}(i_2,j) l^{x'}(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\}
\end{array} \tag{3.10}$$

Observe that $k^x(i_2, j) = k^{x'}(i_1, j)$, $k^x(i_1, j) = k^{x'}(i_2, j)$ and $l^x(i_2, j) = l^{x'}(i_1, j)$, $l^x(i_1, j) = l^{x'}(i_2, j)$ for $j \in J \setminus \{j_1, j_2\}$. Writing (3.10) in terms of point x, we derive

$$\begin{array}{rcl}
a_{i_1 j_1 bp} + a_{i_1 j_2 cq} + \sum\{a_{i_1 j k^x(i_1,j) l^x(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 dr} + a_{i_2 j_2 es} + \sum\{a_{i_2 j k^x(i_2,j) l^x(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\} = \\
a_{i_1 j_1 dr} + a_{i_1 j_2 es} + \sum\{a_{i_1 j k^x(i_2,j) l^x(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
+a_{i_2 j_1 bp} + a_{i_2 j_2 cq} + \sum\{a_{i_2 j k^x(i_1,j) l^x(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\}
\end{array} \tag{3.11}$$

Similarly, since $\bar{x}, \bar{x}' \in P_I$ we have $a\bar{x} = a\bar{x}'$. Terms a_{ijkl}, for every $i \in I \setminus \{i_1, i_2\}$, are again canceled out, so $a\bar{x} = a\bar{x}'$ becomes

$$\begin{array}{rcl}
a_{i_1 j_1 cq} + a_{i_1 j_2 bp} + \sum\{a_{i_1 j k^{\bar{x}}(i_1,j) l^{\bar{x}}(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 es} + a_{i_2 j_2 dr} + \sum\{a_{i_2 j k^{\bar{x}}(i_2,j) l^{\bar{x}}(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\} = \\
a_{i_1 j_1 es} + a_{i_1 j_2 dr} + \sum\{a_{i_1 j k^{\bar{x}'}(i_1,j) l^{\bar{x}'}(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 cq} + a_{i_2 j_2 bp} + \sum\{a_{i_2 j k^{\bar{x}'}(i_2,j) l^{\bar{x}'}(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\}
\end{array} \tag{3.12}$$

Observe that $k^{\bar{x}}(i_2, j) = k^{\bar{x}'}(i_1, j)$, $k^{\bar{x}}(i_1, j) = k^{\bar{x}'}(i_2, j)$ and $l^{\bar{x}}(i_2, j) = l^{\bar{x}'}(i_1, j)$, $l^{\bar{x}}(i_1, j) = l^{\bar{x}'}(i_2, j)$ for $j \in J \setminus \{j_1, j_2\}$. Writing (3.12) in terms of point $\bar{x}$, we derive

$$\begin{array}{rcl}
a_{i_1 j_1 cq} + a_{i_1 j_2 bp} + \sum\{a_{i_1 j k^{\bar{x}}(i_1,j) l^{\bar{x}}(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 es} + a_{i_2 j_2 dr} + \sum\{a_{i_2 j k^{\bar{x}}(i_2,j) l^{\bar{x}}(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\} = \\
a_{i_1 j_1 es} + a_{i_1 j_2 dr} + \sum\{a_{i_1 j k^{\bar{x}}(i_2,j) l^{\bar{x}}(i_2,j)} & : & j \in J \setminus \{j_1, j_2\}\} + \\
a_{i_2 j_1 cq} + a_{i_2 j_2 bp} + \sum\{a_{i_2 j k^{\bar{x}}(i_1,j) l^{\bar{x}}(i_1,j)} & : & j \in J \setminus \{j_1, j_2\}\}
\end{array} \tag{3.13}$$

Subtracting (3.11) from (3.13) and observing that for $i \in \{i_1, i_2\}$ and $j \in J \setminus \{j_1, j_2\}$ we have $k^x(i, j) = k^{\bar{x}}(i, j)$ and $l^x(i, j) = l^{\bar{x}}(i, j)$ (all columns except for j_1, j_2 are common to the OLS pairs represented by points x and $\bar{x}$), we obtain

$$\begin{aligned}
& a_{i_1 j_1 bp} + a_{i_1 j_2 cq} + a_{i_2 j_1 dr} + a_{i_2 j_2 es} - (a_{i_1 j_1 cq} + a_{i_1 j_2 bp} + a_{i_2 j_1 es} + a_{i_2 j_2 dr}) \\
= \; & a_{i_1 j_1 dr} + a_{i_1 j_2 es} + a_{i_2 j_1 bp} + a_{i_2 j_2 cq} - (a_{i_1 j_1 es} + a_{i_1 j_2 dr} + a_{i_2 j_1 cq} + a_{i_2 j_2 bp})
\end{aligned}$$

Table 3.12: Point x_2(Theorem 3.5)

	1	$\cdots$	j_0	$\cdots$
1	k_0		k_1	
$\vdots$				
i_0	k_2		k_3	
$\vdots$				

$\cdots$

	1	$\cdots$	j_0	$\cdots$
1	1		l_1	
$\vdots$				
i_0	l_2		l_3	
$\vdots$				

If we eliminate the negative sign by moving terms in brackets to the other side of the equation and write the elements of sets K and L using the notation $k(i,j)$ and $l(i,j)$ respectively, we obtain equation (3.9). ■

In Proposition 3.6, the role of the sets I, J for the row and column set, respectively, is purely conventional. Any pair of sets from I, J, K, L can be used for the role of row/column set. Hence, for the rest of the paper, the notation $x((m_1, m_2)_{t_1}; (n_1, n_2)_{t_2})$ implies equation (3.9), derived by applying Proposition 3.6 at point x, for rows m_1, m_2 and columns n_1, n_2. In this expression, the first pair denotes the rows whereas the second denotes the columns. The subscripts t_1, t_2 declare the sets that index the rows and the columns, respectively. Following the same convention as for the interchanges, 1 is used to denote set I, 2 is used to denote set J and so on. For example, $x((1, i_1)_1; (1, n)_2)$ denotes equation (3.9) written for rows $1, i_1$ and columns $1, n$ at point x, where elements of the first pair belong to set I and of the second pair to set J.

(Back to the proof of Theorem 3.5). We will show (3.4),...,(3.8) for $i = i_0$, $j = j_0$, $k = k_0$, $l = l_0$, $i_0, j_0, k_0 \neq 1$.

Let $x_1 = x_0$ (Table 3.6). Equation $x_1((1, i_0)_1; (1, j_0)_2)$ is written as:

$$\begin{aligned} &a_{1111} + a_{11k_3l_3} + a_{i_01k_1l_1} + a_{i_01k_2l_2} + a_{1j_0k_1l_1} + a_{1j_0k_2l_2} + a_{i_0j_011} + a_{i_0j_0k_3l_3} \\ &= a_{11k_1l_1} + a_{11k_2l_2} + a_{i_0111} + a_{i_01k_3l_3} + a_{1j_011} + a_{1j_0k_3l_3} + a_{i_0j_0k_1l_1} + a_{i_0j_0k_2l_2} \end{aligned} \tag{3.14}$$

Because of the fact that the two squares must be orthogonal, at most one of values k_3, l_3 can be equal to 1, since pair $(1,1)$ appears already at position $(1,1)$. If $k_3 \neq 1$, derive point $x_2 = x_1(1 \leftrightarrow k_0)_3$. If $k_3 = 1$, it must be that $l_3 \neq 1$. In the latter case, consider point $x_1^* = x_1(K \leftrightarrow L)$, thus enforcing $k_3 \neq 1$ at point x_1^*, and derive $x_2 = x_1^*(1 \leftrightarrow k_0)_3$. Point x_2 is illustrated in Table 3.12.

Equation $x_2((1, i_0)_1; (1, j_0)_2)$ is written as:

$$\begin{aligned} &a_{11k_01} + a_{11k_3l_3} + a_{i_01k_1l_1} + a_{i_01k_2l_2} + a_{1j_0k_1l_1} + a_{1j_0k_2l_2} + a_{i_0j_0k_01} + a_{i_0j_0k_3l_3} \\ &= a_{11k_1l_1} + a_{11k_2l_2} + a_{i_01k_01} + a_{i_01k_3l_3} + a_{1j_0k_01} + a_{1j_0k_3l_3} + a_{i_0j_0k_1l_1} + a_{i_0j_0k_2l_2} \end{aligned} \tag{3.15}$$

Subtracting (3.15) from (3.14) and cancelling identical terms results in

$$a_{1111} - a_{11k_01} + a_{i_0j_011} - a_{i_0j_0k_01} = a_{i_0111} - a_{i_01k_01} + a_{1j_011} - a_{1j_0k_01}$$

which is equivalent to (3.5). Equations (3.6)-(3.8) follow by symmetry.

To show eq. (3.4), consider point x_2, as illustrated in Table 3.12. Since we have already assumed $k_3 \neq 1$, value l_3 can be still equal to 1. If $l_3 \neq 1$, derive point $x_3 = x_2(1 \leftrightarrow l_0)_4$. If $l_3 = 1$, for $n \geq 3$ there exists a column $j_1 \in J \backslash \{1, j_0\}$, such that $l(i_0, j_1)_{x_2} \neq 1$. Setting $x_2^* = x_2(j_0 \leftrightarrow j_1)_2$, results in a point of the form depicted in Table 3.12, having $l(i_0, j_0)_{x_2^*} \neq 1$. Therefore, point x_3 can be now

Table 3.13: Point x_3(Theorem 3.5)

	1	$\cdots$	j_0	$\cdots$
1	k_0		k_1	
$\vdots$				
i_0	k_2		k_3	
$\vdots$				

$\cdots$

	1	$\cdots$	j_0	$\cdots$
1	l_0		l_1	
$\vdots$				
i_0	l_2		l_3	
$\vdots$				

derived as $x_3 = x_2^*(1 \leftrightarrow l_0)_4$. Point x_3 is depicted in Table 3.13.

Equation $x_3((1,i_0)_1;(1,j_0)_2)$ is written as:

$$\begin{aligned} & a_{11k_0l_0} + a_{11k_3l_3} + a_{i_01k_1l_1} + a_{i_01k_2l_2} + a_{1j_0k_1l_1} + a_{1j_0k_2l_2} + a_{i_0j_0k_0l_0} + a_{i_0j_0k_3l_3} \\ &= a_{11k_1l_1} + a_{11k_2l_2} + a_{i_01k_0l_0} + a_{i_01k_3l_3} + a_{1j_0k_0l_0} + a_{1j_0k_3l_3} + a_{i_0j_0k_1l_1} + a_{i_0j_0k_2l_2} \end{aligned} \tag{3.16}$$

Subtracting (3.16) from (3.15) and cancelling identical terms results in

$$a_{11k_01} - a_{11k_0l_0} + a_{i_0j_0k_01} - a_{i_0j_0k_0l_0} = a_{i_01k_01} - a_{i_01k_0l_0} + a_{1j_0k_01} - a_{1j_0k_0l_0}$$

or

$$a_{i_0j_0k_0l_0} = a_{i_0j_0k_01} + a_{i_01k_0l_0} + a_{1j_0k_0l_0} - a_{i_01k_0l_0} - a_{1j_0k_01} - a_{11k_0l_0} + a_{11k_01} \tag{3.17}$$

Substituting in (3.17) terms from equations (3.5)-(3.8) results in equation (3.4).

Finally, (3.3) is true since for $n \geq 3$ and different than 6, $P_I^n \neq \emptyset$ (see [60, Theorem 2.9]). This implies that there exists at least one 0-1 vector x, for which (2.1),...,(2.6) are satisfied. Hence, by multiplying these equations with the corresponding scalars and summing over all rows we get:

$$\begin{aligned} ax &= \sum\{\lambda_{kl}^1 : k \in K, l \in L\} + \sum\{\lambda_{il}^2 : i \in I, l \in L\} \\ &+ \sum\{\lambda_{jl}^3 : j \in J, l \in L\} + \sum\{\lambda_{ij}^4 : i \in I, j \in J\} \\ &+ \sum\{\lambda_{jk}^5 : j \in J, k \in K\} + \sum\{\lambda_{ik}^6 : i \in I, k \in K\} \end{aligned}$$

This completes the proof for general n. The theorem holds for $n \geq 4$ and $n \neq 6$ since the proof requires at most four distinct values for each index. ■

Corollary 3.7 *For $n \geq 4$ and $n \neq 6$, dim$(P_I) = (n-1)^4 + 4 \cdot (n-1)^3$.*

Proof. Exactly as in Corollary 3.3. ■

3.3 Dimension of the inequalities defining P_L

In this section, we examine which of the constraints defining P_L are facet defining for P_I.

Proposition 3.8 *For $n \geq 4$ and $n \neq 6$, every inequality of the type $x_c \geq 0$ for $c \in C$ defines a facet of P_I.*

Proof. For any $c \in C$ consider the polytope $P_I^c = \{x \in P_I : x_c = 0\}$. We need to show that $\dim(P_I^c) = \dim(P_I) - 1$. Evidently, $\dim(P_I^c) \leq n^4 - 1 - \text{rank}(A^c)$ where A^c is the matrix obtained from

A if we remove column a^c. It is not hard to see that the rank of A^c is equal to the rank of A. This is immediate, if the column a^c is not among the columns of the upper triangular matrix described in Theorem 3.2; otherwise it follows by symmetry. Therefore, $\dim(P_I^c) \leq n^4 - 6n^2 + 8n - 4$. To prove that this bound is attained, we use the same approach as in the proof of Theorem 3.5, i.e. show that any equation $ax = a_0$ (different than $x_c = 0$) satisfied for every $x \in P_I^c$ is a linear combination of $A^c x = e$. The proof goes through essentially unchanged. ■

Proposition 3.9 *For $n \geq 4$ and $n \neq 6$, every inequality $x_c \leq 1$ for $c \in C$ does not define a facet of P_I*

Proof. For any $c \in C$ consider the polytope $P_I^c = \{x \in P_I : x_c = 1\}$. We will show that $\dim(P_I^c) < \dim(P_I) - 1$. We know that $\dim(P_I^c) \leq \dim(P_L^c)$ where P_L^c is the linear relaxation of P_I^c. Setting x_c to one is equivalent to setting the variables belonging to the common constraints with x_c to zero. The number of these variables is $2(3n-1)(n-1)$ (Prop. 2.1). Thus, $\dim(P_L^c) = n^4 - 2(3n-1)(n-1) - \text{rank}(A_n^c)$ where A_n^c is the matrix obtained from A by removing the columns corresponding to the variables set to zero. Obviously $\text{rank}(A_{n-1}) \leq \text{rank}(A_n^c)$ where A_{n-1} is the constraint matrix of the *OLS* of order $n-1$. By Theorem 3.2 $\text{rank}(A_{n-1}) = 6(n-1)^2 - 8(n-1) + 3$ so

$$\begin{aligned}\dim(P_I^c) \leq \dim(P_L^c) &\leq n^4 - 2(3n-1)(n-1) - (6(n-1)^2 - 8(n-1) + 3)\\ &= n^4 - 12n^2 + 28n - 19\end{aligned}$$

which is less than $\dim(P_I) - 1 = n^4 - 6n^2 + 8n - 4$ for $n \geq 3$. ■

It is easy to see that cliques of type I do not induce facets of P_I. For each of these cliques, the coefficient vector of the corresponding inequality is identical to a row of the A matrix. Thus, each of this inequalities is satisfied as equality by all $x \in P_I$, therefore defining an improper face of P_I.

3.4 Clique inequalities

If $Q \subset C$, is the node set of a clique in the graph $G_A = (C, E_C)$, then the complete sub-graph $K_{|Q|}$ induces the inequality

$$\sum \{x_d : d \in Q\} \leq 1 \tag{3.18}$$

Inequalities (3.18) define facets of the general set-packing polytope P_{SPP}. Therefore, they also induce facets of the OLS packing polytope $\tilde{P}_I$. The proof is trivial and can be found, among others, in ([69]). However, no similar result has been proven for general set-partitioning polytope P_{SP}, which is itself a face of P_{SPP}. In particular, no trivial argument has been found, proving that inequalities (3.18) are facet-defining for the OLS polytope P_I, which is a face of $\tilde{P}_I$.

Cliques of type II give rise to inequalities of the form:

$$\sum \{x_d : d \in Q(c)\} \leq 1 \tag{3.19}$$

for every $c \in C$.

Cliques of type III give rise to inequalities of the form:

$$\sum \{x_d : d \in Q(c, s)\} \leq 1 \tag{3.20}$$

for every $c, s \in C$ such that $\mid c \cap s \mid = 1$.

3.4.1 Dimension and Chvátal rank of clique inequalities

In [2], it has been proved that the inequalities induced by cliques of both types II & III define facets of P_I. We reproduce here the main results.

Theorem 3.10 *([2, Theorem 4.11]) Let $Q(c)$ denote the node set of a clique of type II. Then, for $n \geq 5$ and $n \neq 6$ the inequality*

$$\sum\{x_q : q \in Q(c)\} \leq 1 \tag{3.21}$$

defines a facet of P_I for every $c \in C$.

Theorem 3.11 *([2, Theorem 4.13]) Let $Q(c, s)$ denote the node set of a clique of type III. Then, for $n \geq 5$ and $n \neq 6$ the inequality*

$$\sum\{x_q : Q(c, s)\} \leq 1 \tag{3.22}$$

defines a facet of P_I for every $c, s \in C$ such that $\mid c \cap s \mid = 1$.

Let us now examine the Chvátal rank of clique inequalities.

Proposition 3.12 *The inequalities (3.19) are of Chvátal rank 2.*

Proof. We will first derive a lower bound for the rank of (3.19), by showing that the inequality cannot be of rank 1. We will proceed by illustrating how the inequality can be derived within two steps of the Chvátal-Gomory procedure. This will provide an upper bound of 2 on its rank.

Assume w.l.o.g. $c = (i_0, j_0, k_0, l_0)$. The node set of the clique is $Q(c) = \{(i_0, j_0, k_0, l_0), \{(i, j_0, k_0, l_0) : i \in I\backslash\{i_0\}\}, \{(i_0, j, k_0, l_0) : j \in J\backslash\{j_0\}\}, \{(i_0, j_0, k, l_0) : k \in K\backslash\{k_0\}\}, \{(i_0, j_0, k_0, l) : l \in L\backslash\{l_0\}\}\}$. The induced inequality is:

$$x_{i_0j_0k_0l_0} + \sum_{i \in I\backslash\{i_0\}} x_{ij_0k_0l_0} + \sum_{j \in J\backslash\{j_0\}} x_{i_0jk_0l_0} + \sum_{k \in K\backslash\{k_0\}} x_{i_0j_0kl_0} + \sum_{l \in L\backslash\{l_0\}} x_{i_0j_0k_0l} \leq 1 \tag{3.23}$$

If the inequality (3.23) is of Chvátal rank 1, there exists $0 < \varepsilon < 1$, such that every solution to the LP-relaxation of OLS, i.e. $Ax = e, x \geq 0$, satisfies:

$$x_{i_0j_0k_0l_0} + \sum_{i \in I\backslash\{i_0\}} x_{ij_0k_0l_0} + \sum_{j \in J\backslash\{j_0\}} x_{i_0jk_0l_0} + \sum_{k \in K\backslash\{k_0\}} x_{i_0j_0kl_0} + \sum_{l \in L\backslash\{l_0\}} x_{i_0j_0k_0l} \leq 2 - \varepsilon \tag{3.24}$$

A solution having $x_{ij_0k_0l_0} = x_{i_0jk_0l_0} = x_{i_0j_0kl_0} = x_{i_0j_0k_0l} = \frac{1}{2(n-1)}$, $i \in I\backslash\{i_0\}$, $j \in J\backslash\{j_0\}$, $k \in K\backslash\{k_0\}$, $L \in L\backslash\{l_0\}$, and $x_{i_0j_0k_0l_0} = 0$, violates (3.24), since the its l.h.s. has $4(n-1)$ variables equal to $\frac{1}{2(n-1)}$, therefore becomes equal to 2. Such a solution also has:

- $x_{ijk_0l_0} = x_{ij_0kl_0} = x_{ij_0k_0l} = x_{i_0jkl_0} = x_{i_0jk_0l} = x_{i_0j_0kl} = 0$,
- $x_{i_0jkl} = x_{ij_0kl} = x_{ijk_0l} = x_{ijkl_0} = \frac{2n-3}{2(n-1)^3}$,
- $x_{ijkl} = \frac{(n-2)^2}{(n-1)^4}$

for all $i \in I\backslash\{i_0\}$, $j \in J\backslash\{j_0\}$, $k \in K\backslash\{k_0\}$, $L \in L\backslash\{l_0\}$.

To see that $x \in P_L$, assume row $(m_1, m_2) \in M_1 \times M_2$, $M_1, M_2 = I, J, K, L$, $M_1 \neq M_2$ and $c = (i_0, j_0, k_0, l_0)$. Then:

Case 3.12.1 *If $m_1, m_2 \subset c$, the row has $2(n-1)$ variables equal to $\frac{1}{2(n-1)}$ and $2n-1$ variables equal to 0.*

Case 3.12.2 *If $m_1 \subset c, m_2 \subsetneq c$ or $m_1 \subsetneq c, m_2 \subset c$, the row has 1 variable equal to $\frac{1}{2(n-1)}$, $(n-1)^2$ variables equal to $\frac{2n-3}{2(n-1)^3}$ and $2(n-1)$ variables equal to 0.*

Case 3.12.3 *If $m_1, m_2 \subsetneq c$, the row has $2(n-1)$ variables equal to $\frac{2n-3}{2(n-1)^3}$, $(n-1)^2$ variables equal to $\frac{(n-2)^2}{(n-1)^4}$ and 1 variable equal to 0.*

Therefore, inequality (3.23) is of Chvátal rank at least 2.

On the other hand, adding rows (i_0, l_0), (j_0, l_0), (k_0, l_0), dividing the resulting inequality by 2 and rounding down both sides gives:

$$x_{i_0j_0k_0l_0} + \sum_{i\in I\setminus\{i_0\}} x_{ij_0k_0l_0} + \sum_{j\in J\setminus\{j_0\}} x_{i_0jk_0l_0} + \sum_{k\in K\setminus\{k_0\}} x_{i_0j_0kl_0} \leq 1 \tag{3.25}$$

Adding rows (i_0, k_0), (j_0, k_0), (k_0, l_0), dividing the resulting inequality by 2 and rounding down both sides gives:

$$x_{i_0j_0k_0l_0} + \sum_{i\in I\setminus\{i_0\}} x_{ij_0k_0l_0} + \sum_{j\in J\setminus\{j_0\}} x_{i_0jk_0l_0} + \sum_{l\in L\setminus\{l_0\}} x_{i_0j_0k_0l} \leq 1 \tag{3.26}$$

Adding rows (i_0, j_0), (j_0, k_0), (j_0, l_0), dividing the resulting inequality by 2 and rounding down both sides gives:

$$x_{i_0j_0k_0l_0} + \sum_{i\in I\setminus\{i_0\}} x_{ij_0k_0l_0} + \sum_{k\in K\setminus\{k_0\}} x_{i_0j_0kl_0} + \sum_{l\in L\setminus\{l_0\}} x_{i_0j_0k_0l} \leq 1 \tag{3.27}$$

Adding rows (i_0, j_0), (i_0, k_0), (i_0, l_0), dividing the resulting inequality by 2 and rounding down both sides gives:

$$x_{i_0j_0k_0l_0} + \sum_{j\in J\setminus\{j_0\}} x_{i_0jk_0l_0} + \sum_{k\in K\setminus\{k_0\}} x_{i_0j_0kl_0} + \sum_{l\in L\setminus\{l_0\}} x_{i_0j_0k_0l} \leq 1 \tag{3.28}$$

Finally, adding rank 1 inequalities (3.25)-(3.28) gives the following inequality:

$$4x_{i_0j_0k_0l_0} + 3\sum_{i\in I\setminus\{i_0\}} x_{ij_0k_0l_0} + 3\sum_{j\in J\setminus\{j_0\}} x_{i_0jk_0l_0} + 3\sum_{k\in K\setminus\{k_0\}} x_{i_0j_0kl_0} + 3\sum_{l\in L\setminus\{l_0\}} x_{i_0j_0k_0l} \leq 1 \tag{3.29}$$

Dividing inequality (3.29) by 3 and rounding down both sides gives inequality (3.23). Therefore, inequality (3.23) is of rank at most 2. The proof is complete. ■

Proposition 3.13 *The inequalities (3.20) are of Chvátal rank 2.*

Proof. The proof will essentially proceed in the same way as the proof of Proposition 3.12.

Assume w.l.o.g. $c = (i_0, j_0, k_0, l_0)$ and $s = (i_0, j_1, k_1, l_1)$. The node set of the clique is $Q(c,s) = \{(i_0, j_0, k_0, l_0), (i_0, j_0, k_1, l_1), (i_0, j_1, k_0, l_1), (i_0, j_1, k_1, l_0)\}$, the induced inequality being:

$$x_{i_0j_0k_0l_0} + x_{i_0j_0k_1l_1} + x_{i_0j_1k_0l_1} + x_{i_0j_1k_1l_0} \leq 1. \tag{3.30}$$

If the inequality (3.30) is of Chvátal rank 1, then there exists $0 < \varepsilon < 1$, such that every solution to the LP-relaxation of OLS, i.e. $Ax = e, x \geq 0$, satisfies:

$$x_{i_0j_0k_0l_0} + x_{i_0j_0k_1l_1} + x_{i_0j_1k_0l_1} + x_{i_0j_1k_1l_0} \leq 2 - \varepsilon \tag{3.31}$$

However, any solution having $x_{i_0j_0k_0l_0} = x_{i_0j_0k_1l_1} = x_{i_0j_1k_0l_1} = x_{i_0j_1k_1l_0} = \frac{1}{2}$ violates (3.31). We will show that such a solution always exists for $n \geq 3$, distinguishing between the cases where n is even and n is odd, thus proving that the rank of the inequality (3.30) is at least 2.

If n is even, then for each $(i_v, j_x) \in I \times J$ define:

$$y_1 = \left(\left\lceil\frac{v}{2}\right\rceil \cdot 2 + \left\lfloor\frac{x}{2}\right\rfloor \cdot 2\right) \bmod n, z_1 = \left(y_1 + \left\lfloor\frac{v}{2}\right\rfloor \cdot 2 + x - \left\lfloor\frac{x}{2}\right\rfloor \cdot 2\right) \bmod n$$

and

$$y_2 = \left(\left\lceil\frac{v}{2}\right\rceil \cdot 2 + \left\lfloor\frac{x}{2}\right\rfloor \cdot 2 + 1\right) \bmod n, z_2 = \left(y_2 + \left\lfloor\frac{v}{2}\right\rfloor \cdot 2 - x + \left\lfloor\frac{x}{2}\right\rfloor \cdot 2\right) \bmod n$$

The solution must satisfy:

$$x_{i_vj_xk_yl_z} = \left\{ \begin{array}{ll} \frac{1}{2}, & \text{if } y = y_1, z = z_1 \text{ or } y = y_2, z = z_2 \\ 0, & \text{otherwise} \end{array} \right\}$$

for all $(i_v, j_x) \in I \times J$. It is easy to see that $v, x, y, z \in \{0, 1, ..., n-1\}$.

To illustrate this solution with respect to an *OLS* structure, it is convenient to assume that $I = J = K = L = \{0, 1, ..., n-1\}$, n even. I is the row set, J is the column set and K, L are the value sets for the first and the second Latin square, respectively. For each cell $(i, j) \in I \times J$, define:

$$k_1 = \left(\left\lceil\frac{i}{2}\right\rceil \cdot 2 + \left\lfloor\frac{j}{2}\right\rfloor \cdot 2\right) \bmod n, l_1 = \left(k_1 + \left\lfloor\frac{i}{2}\right\rfloor \cdot 2 + j - \left\lfloor\frac{j}{2}\right\rfloor \cdot 2\right) \bmod n$$

and

$$k_2 = \left(\left\lceil\frac{i}{2}\right\rceil \cdot 2 + \left\lfloor\frac{j}{2}\right\rfloor \cdot 2 + 1\right) \bmod n, l_2 = \left(k_2 + \left\lfloor\frac{i}{2}\right\rfloor \cdot 2 - j + \left\lfloor\frac{j}{2}\right\rfloor \cdot 2\right) \bmod n$$

The solution must satisfy:

$$x_{ijkl} = \left\{ \begin{array}{ll} \frac{1}{2}, & \text{if } k = k_1, l = l_1 \text{ or } k = k_2, l = l_2 \\ 0, & \text{otherwise} \end{array} \right\}$$

The non-zero variables, for $n = 6$, are depicted in Figure 3.3. Pair (k, l) placed at cell (i, j) implies that $x_{ijkl} = \frac{1}{2}$. The row values in the left margin are the values of index i, whereas the marginal column values are the ones of index j. It is easy to verify that exactly two variables at the l.h.s. of each constraint are set to $\frac{1}{2}$ by checking that each value of index k (or l) appears exactly twice in each row/column, each pair (k, l) appears at exactly two cells and each cell has exactly two non-zero variables.

If n is odd, the solution does not follow a pattern as concrete as above, therefore it can be better described in the format used in Figure 3.3. Assume again w.l.o.g. that $I = J = K = L = \{0, 1, ..., n-1\}$, n odd and construct a square matrix, with the marginal values in rows and columns corresponding to elements of sets I and J, respectively. The matrix is symmetric, with respect to the main bottom-left to top-right diagonal (i.e. the diagonal $(n-1, 0)$, $(n-2, 1)$,...., $(0, n-1)$), therefore only the upper half will be described. Figure 3.4 exhibits the structure of the matrix.

The non-zero x_{0jkl} variables (i.e. the ones in the first row) are fixed exactly as in the case of n being even, except for variable $x_{0(n-1)(n-1)(n-1)}$, the single variable set to 1. The non-zero $x_{i(n-1)kl}$

	0	1	2	3	4	5
0	00,11	01,10	22,33	23,32	44,55	45,54
1	22,33	23,32	44,55	45,54	00,11	01,10
2	24,35	25,34	40,51,	41,50	02,13	03,12
3	40,51	41,50	02,13	03,12	24,35	25,34
4	42,53	43,52	04,15	05,14	20,31	21,30
5	04,15	05,14	20,31	21,30	42,53	43,52

Figure 3.3: A solution violating (3.31) for $n = 6$.

	0	1			n-2	n-1	n-1
0	00,11	01,10	...	...	(n-3)(n-3),(n-2)(n-2)	(n-3)(n-2),(n-2)(n-3)	(n-1)(n-1)
1	(n-5)(n-3),(n-4)(n-2)		...	...	(n-1)0,(n-1)1		(n-3)(n-2),(n-2)(n-3))
2							(n-3)(n-3),(n-2)(n-2)
...	...	...	...	...	...	...	...
n-2	(n-1)2,(n-1)3		...	...			01,10
n-1						(n-5)(n-3),(n-4)(n-2)	00,11

Figure 3.4: A solution violating (3.31) for n odd

variables (i.e. the ones in the last column) are fixed in the manner posed by the above mentioned symmetry. Excluding the first row and the last column, we are left with an $(n-1) \times (n-1)$ square submatrix (of even size), which can be split into $\frac{(n-1)^2}{4}$ 2×2 submatrices.

Each such 2×2 submatrix, which is "above" the main diagonal, involves two pairs of consecutive indices from each of the sets K, L according to the following pattern (remember that all 4 sets contain elements $\{0, 1, ..., n-1\}$) :

$$\begin{array}{|c|c|} \hline yz, (y+1)(z+1) & zy, (z+1)(y+1) \\ \hline z(y+1), (z+1)y & y(z+1), (y+1)z \\ \hline \end{array} \tag{3.32}$$

where $y, z = 0, 2, ..., n-3$.

It is therefore enough to define only the upper left element of each such submatrix, appearing at rows $1, 3, .., n-4$ and columns $0, 2, ..., n-5$. At row i and column $j(i \leq n-4$ and odd, $j \leq n-5$ and even, $i + j \leq n-4)$, there must be $y = n - (i+j) - 4$ and $z = y + n - i$. If $z > n-3$, z is replaced by $z \bmod (n-3)$.

The remaining 2×2 submatrices are the last to be filled according to the pattern:

$$\begin{array}{|c|c|} \hline (n-1)y, (n-1)(y+1) & y(n-1), (y+1)(n-1) \\ \hline y(n-1), (y+1)(n-1) & (n-1)y, (n-1)(y+1) \\ \hline \end{array} \tag{3.33}$$

where $y = 0, 2, ..., n-3$. Note that the top-right and bottom left cells are on the diagonal $(n-1, 0)$, $(n-2, 1), ...,$ $(0, n-1)$ (i.e. the main bottom-left to top-right diagonal). The value of y must be the one appearing only once in that column, after all other cells have been filled. Figure 3.5 is also an example for $n = 7$. It is easy to check that exactly two variables at the l.h.s. of each constraint are set to $\frac{1}{2}$ or exactly one variable (i.e. x_{0666}) is set to 1.

We have shown that the rank of (3.30) is at least two. Next we will show that the rank is at most 2, by deriving (3.30) as a linear combination of rank1 inequalities. Adding the rows

	0	1	2	3	4	5	6
0	00,11	01,10	22,33	23,32	44,55	45,54	66
1	24,35	42,53	02,13	20,31	60,61	06,16	45,54
2	43,52	25,34	21,30	03,12	06,16	60,61	44,55
3	04,15	40,51	64,65	46,56	03,12	20,31	23,32
4	50,41	05,14	46,56	64,65	21,30	02,13	22,33
5	62,63	26,36	05,14	40,51	25,34	42,53	01,10
6	26,36	62,63	50,41	04,15	43,52	24,35	00,11

Figure 3.5: A solution violating (3.31) for $n = 7$

$(i_0, j_0), (i_0, k_0), (i_0, l_1)$, each one weighted by $\frac{1}{2}$, gives an inequality, where variables $x_{i_0 j_0 k_0 l_0}, x_{i_0 j_0 k_1 l_1}$ & $x_{i_0 j_1 k_0 l_1}$ appear with coefficient 1, variable $x_{i_0 j_0 k_0 l_1}$ appears with coefficient $\frac{3}{2}$ and all other variables appear with coefficient $\frac{1}{2}$. The r.h.s. is $\frac{3}{2}$. Rounding down both sides results in the inequality:

$$x_{i_0 j_0 k_0 l_0} + x_{i_0 j_0 k_1 l_1} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_0 k_0 l_1} \leq 1 \tag{3.34}$$

Applying the same procedure to rows $(i_0, j_0), (i_0, k_1), (i_0, l_0)$ results in the the inequality:

$$x_{i_0 j_0 k_0 l_0} + x_{i_0 j_0 k_1 l_1} + x_{i_0 j_1 k_1 l_0} + x_{i_0 j_0 k_1 l_0} \leq 1 \tag{3.35}$$

Applying the same procedure to rows $(i_0, j_1), (i_0, k_0), (i_0, l_0)$ results in the the inequality:

$$x_{i_0 j_0 k_0 l_0} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_1 k_1 l_0} + x_{i_0 j_1 k_0 l_0} \leq 1 \tag{3.36}$$

Applying the same procedure to rows $(i_0, j_1), (i_0, k_1), (i_0, l_1)$ results in the the inequality:

$$x_{i_0 j_0 k_1 l_1} + x_{i_0 j_1 k_0 l_1} + x_{i_0 j_1 k_1 l_0} + x_{i_0 j_1 k_1 l_1} \leq 1 \tag{3.37}$$

Adding rank 1 inequalities (3.34)-(3.37) gives the inequality:

$$3(x_{i_0 j_0 k_0 l_0} + x_{i_0 j_0 k_1 l_1} + x_{i_0 j_1 k_1 l_0} + x_{i_0 j_1 k_0 l_1}) + (x_{i_0 j_0 k_0 l_1} + x_{i_0 j_0 k_1 l_0} + x_{i_0 j_1 k_0 l_0} + x_{i_0 j_1 k_1 l_1}) \leq 4 \tag{3.38}$$

Dividing inequality (3.38) by 3 and rounding down both sides gives inequality (3.30). This implies that inequality (3.30) is of rank at most 2 and the proof is complete. ■

3.4.2 Separation algorithms for clique inequalities

Facet-defining inequalities are of great importance since they describe the convex hull of integer solutions for a problem. Therefore, if we knew all facet-defining inequalities of an integer polytope, we would be able to solve the integer problem by incorporating them into the constraint matrix and then solving the linear programming relaxation. In practice, however, this is not easy, since for most problems *a)* not all the facets of the underlying convex hull of integer points are known, and *b)* the number of facets is not polynomially bounded on the size of the problem, therefore resulting in a constraint matrix of exponential size. For these reasons, most algorithms consider the known facet inequalities only when they are violated by a certain point of the linear relaxation polytope. Although determining whether an arbitrary non-integer solution violates a facet-defining inequality of the convex hull of integer solutions for an $\mathcal{NP}$-hard problem is generally also $\mathcal{NP}$-hard, it is sometimes possible to do that efficiently for certain classes of facets. In particular, with respect to the *OLS*, we propose two polynomial procedures, one for each class of facet defining cliques, that deal with the problem of detecting a violated clique inequality.

The separation algorithms of this paragraph have been motivated by the ones in [9]. First, we provide an algorithm that detects a violated facet-defining inequality induced by cliques of type II.

Algorithm 3.14 (Separation of Cliques of type II) *Let $x \in P_L \backslash P_I$ and $v \in \mathbb{N}$ such that $v \geq 5$.*

Step I *Set $d_c = 0$, for all $c \in C$.*

Step 2: *For all $s \in C$ check x_s. If $x_s \geq \frac{1}{vn}$, set $d_c = d_c + x_s$ for all $c \in Q(s)$. If $d_c > 1$ stop: the inequality $\sum\{x_q : q \in Q(c)\} \leq 1$ is violated; otherwise, continue.*

Step 3: *For all $c \in C$ if $d_c > \frac{v-4}{v}$, check whether the inequality $\sum\{x_q : q \in Q(c)\} \leq 1$ is violated. If yes stop; otherwise continue.*

In order to prove the correctness and complexity of the algorithm we need a number of intermediate results.

Lemma 3.15 *For a point $x \in P_L$ and a positive integer v, the number of components of x with value $\geq v$ is $\leq \frac{n^2}{v}$.*

Proof. The value of the linear program $L = \max\{ex : x \in P_L\}$ can be easily shown to be n^2 since the vectors $x \in \mathbb{R}^{n^4}$ and $u \in \mathbb{R}^{6n^2}$ defined by $x_c = \frac{1}{n^2}, \forall c \in C$ and $u_r = \frac{1}{6}, \forall r \in R$ are feasible solutions to L and its dual, respectively. Therefore, they are optimal. If more than $\frac{n^2}{v}$ components of x have values greater than or equal to v, the value of ex would be greater than n^2 contradicting the above. ∎

Lemma 3.16 *For any $x \in P_L$ and any positive integer v, the number of $c \in C$, such that $\sum\{x_q : q \in Q(c)\} \geq v$ is $\leq \frac{n^2(4n-3)}{v}$.*

Proof. Consider

$$\sum\{\sum\{x_q : q \in Q(c)\} : c \in C\} \tag{3.39}$$

Recall that $\mid Q(c) \mid = 4n-3$ for all $c \in C$ which implies that each x_c appears $4n-3$ times in (3.39). Hence:

$$\sum\{\sum\{x_q : q \in Q(c)\} : c \in C\} = (4n-3)\sum\{x_c : c \in C\} \leq (4n-3)n^2$$

If there were more than $\frac{n^2(4n-3)}{v}$ $\sum\{x_q : q \in Q(c)\}$ with a value greater than or equal to v then it would be $\sum\{\sum\{x_q : q \in Q(c)\} : c \in C\} > n^2(4n-3)$ contradicting the above. ∎

Theorem 3.17 *Algorithm 3.14 determines in $O(n^4)$ steps whether a given $x \in P_L$ violates a facet-defining inequality of type II.*

Proof. Let us first prove that the algorithm is correct. Assume that the inequality $\sum\{x_q : q \in Q(c)\} \leq 1$ is violated for some $c \in C$. Then

$$\begin{aligned} d_c &= \sum\{x_q : q \in Q(c), x_q \geq \frac{1}{vn}\} > 1 - \sum\{x_q : q \in Q(c), x_q < \frac{1}{vn}\} \\ &\geq 1 - \frac{4n-3}{vn} \geq \frac{v-4}{v} \end{aligned}$$

Hence, violation is detected at *Step* 3 of the algorithm. Therefore the algorithm is correct.

Let us now examine the complexity of the algorithm. At *Step* 1, we initialize n^4 counters. At *Step* 2, there can be at most vn^3 components of a fractional point x, which are examined (Lemma 3.15). For each of those, $4n-3$ counters are updated since there are $4n-3$ nodes in the node set of a clique of type II. Hence, in the worst case the complexity of *Step* 2 is $vn^3(4n-3)$. At *Step* 3, the number of $c \in C$ for which $\sum\{x_q : q \in Q(c)\} > \frac{v-4}{v}$ is at most $\frac{vn^2(4n-3)}{v-4}$ (Lemma 3.16). For each such c we need $4n-3$ extra steps to check whether the corresponding inequality is indeed violated. Hence, the complexity of *Step* 3 is $\frac{vn^2(4n-3)^2}{v-4}$. The overall complexity of the algorithm is

$$f(v,n) = n^4 + vn^3(4n-3) + \frac{vn^2(4n-3)^2}{v-4} \tag{3.40}$$

which is $O(n^4)$.

The value of v that minimizes $f(v,n)$ is found by setting the first derivative with respect to v to zero:

$$\frac{\partial f(v,n)}{\partial v} = n(v-4)^4 - 4(4n-3) = 0 \Rightarrow v = 4 + \sqrt{16 - \frac{12}{n}}$$

which for large n produces $v = 8$. For this value of v eq. (3.40) becomes $f(n) = n^4 + 8n^3(4n-3) + 2n^2(4n-3)^2$. ■

Note that the complexity of the above algorithm remains linear with respect to the number of variables, therefore it is the lowest possible (see also [9]). A separation algorithm for cliques of type III follows.

Algorithm 3.18 (Separation of Cliques of type III) *Let $x \in P_L \backslash P_I$.*

Step 1: *For all $c \in C$, if $\frac{1}{4} < x_c < 1$:*

Step 2: *Then for all $t \in C$ with $\mid c \cap t \mid = 2$, if $x_t > \frac{1-x_c}{3}$*

Step 3: *Then for all $s \in C$, such that $\mid c \cap s \mid = 1$ and $\mid s \cap t \mid = 3$, if $\sum\{x_q : q \in Q(c,s)\} > 1$ stop: a violated inequality has been found; otherwise continue.*

Theorem 3.19 *Algorithm 3.18 determines in $O(n^4)$ steps whether a given $x \in P_L$ violates a facet defining inequality of type III.*

Proof. Let us examine the correctness of the algorithm. $Q(c,s)$ is a node set of a 4-clique thus if a non-integer point x violates $\sum\{x_q : q \in Q(c,s)\} \leq 1$ at least one component of x must be $> \frac{1}{4}$. W.l.o.g. assume that $x_c > \frac{1}{4}$ for $c = (i_0, j_0, k_0, l_0)$. Since $x \in P_L$, variable x_c appears in the following constraints:

$$\sum\{x_{ijk_0l_0} : i \in I, j \in j\} = 1 \tag{3.41}$$

$$\sum\{x_{i_0jkl_0} : j \in J, k \in K\} = 1 \tag{3.42}$$

$$\sum\{x_{ij_0kl_0} : i \in I, k \in K\} = 1 \tag{3.43}$$

$$\sum\{x_{i_0j_0kl} : k \in K, l \in L\} = 1 \tag{3.44}$$

$$\sum\{x_{ij_0k_0l} : i \in I, l \in L\} = 1 \tag{3.45}$$

$$\sum\{x_{i_0jk_0l} : j \in J, l \in L\} = 1 \tag{3.46}$$

If $x_c = 1$ then for all $t \in C$ such that $\mid c \cap t \mid = 2$ we have $x_t = 0$. Hence, the inequality $\sum\{x_q : q \in Q(c,s)\} \leq 1$ is satisfied as equality for all $s \in C$ such that $\mid c \cap s \mid = 1$. Therefore, if x violates such an inequality, it must be that $\frac{1}{4} < x_c < 1$. Since $\mid Q(c,s) \mid = 4$, the condition $x_t > \frac{1-x_c}{3}$ must hold for at least one $t \in Q(c,s) \backslash \{c\}$. Consequently, Algorithm 3.18 is correct.

Concerning the complexity of the algorithm, we note that the comparison in *Step* 1 is executed in the worst case n^4 times, once for each variable. The number of variables with value $> \frac{1}{4}$ is at most $4n^2$ (Lemma 3.15). For each such variable, there are $6(n-1)^2$ $t \in C$ such that $\mid c \cap t \mid = 2$, as indicated by constraints (3.41),...,(3.46). Hence, we need $24n^2(n-1)^2$ comparisons to identify all such ordered pairs (c,t), yielding a complexity of $O(n^4)$. For each such c, the number of t cannot be more than 3 in each of (3.41),...,(3.46), since otherwise one of these inequalities would be violated. Thus the total number of t given c, for which $x_t > \frac{1-x_c}{3}$ is satisfied, is 18. For c, t given, there are at most $n-1$ $s \in C$ such that $\mid c \cap s \mid = 1$. Thus, *Step* 3 will be executed $4n^2 \times 18 \times (n-1)$ times, i.e. its complexity is of the order $O(n^3)$. The total complexity is $O(n^4)$. ■

Corollary 3.20 *Whether there exists a violated clique inequality can be detected in linear time with respect to the number of variables, i.e. in $O(n^4)$ steps.*

3.5 Antiweb inequalities

For c, $d \in C$, $|c \cap d| = 0$, let $A(c,d) = \{f \in C : f \subseteq c \cup d\}$. It has been proved that the inequality

$$X(A(c,d)) = \sum\{x_f : f \in A(c,d)\} \leq 2 \tag{3.47}$$

can be regarded as lifted from an $(8,3)$ antiweb. Recall also its form for $c = (i_0, j_0, k_0, l_0)$, $d = (i_1, j_1, k_1, l_1)$:

$$\begin{array}{ll} x_{i_0j_0k_0l_0} + x_{i_0j_0k_0l_1} + x_{i_0j_0k_1l_0} + x_{i_0j_0k_1l_1} + x_{i_0j_1k_0l_0} + x_{i_0j_1k_0l_1} + x_{i_0j_1k_1l_0} + x_{i_0j_1k_1l_1} & \\ x_{i_1j_0k_0l_0} + x_{i_1j_0k_0l_1} + x_{i_1j_0k_1l_0} + x_{i_1j_0k_1l_1} + x_{i_1j_1k_0l_0} + x_{i_1j_1k_0l_1} + x_{i_1j_1k_1l_0} + x_{i_1j_1k_1l_1} & \leq 2 \end{array} \tag{3.48}$$

Both the Chvátal rank and the dimension of this inequality remain unknown. The fact that this inequality cannot be further lifted, i.e. it cannot be made stronger, suggests that this inequality cannot be dominated. This implies that the face defined by inequality (3.47) is maximal with respect to set inclusion, i.e. is not included in any other face defining polytope P_I. A beneficial feature, however, is that inequalities (3.47) can also be separated in linear time with respect to the number of variables, i.e. in $O(n^4)$ steps, although their number is $O(n^8)$. The following separation algorithm achieves this result. We may assume that the previously exhibited separation algorithms for clique inequalities have already been executed for a specific point $x \in P_L \backslash P_I$ and that no violated clique inequality of type II has been identified. This assumption is possible without loss of generality and is also important for the correctness and complexity of the following algorithm.

Algorithm 3.21 *Let $x \in P_L \backslash P_I$, which violates no clique inequality.*

```
Step I   For all $c \in C$, if $\frac{1}{8} < x_c < 1$, then:
Step II  For all $d \in C$, such that $|c \cap d| = 2$ if $x_d > \frac{1-x_c}{10}$ then:
Step III For all $g \in C$, such that $|c \cap g| = 0, |d \cap g| = 2$, check whether the inequality $X(A(c,g)) \leq 2$ is
         violated. If yes, stop; otherwise, continue.
```

Step IV For all $d \in C$, such that $|c \cap d| = 3$, if $x_d > \frac{1-x_c}{10}$ then:

Step V For all $f \in C$, such that $|c \cap f| = 2, |c \cap f| = 3$, if $x_d > \frac{1-x_c-x_d}{9}$ then:

Step VI For all $g \in C$, such that $|c \cap g| = 0, |f \cap g| = 2$, check whether the inequality $X(A(c,g)) \leq 2$ is violated. If yes, stop; otherwise, continue.

Proposition 3.22 *Algorithm 3.21 determines in $O(n^4)$ steps whether a given $x \in P_L \backslash P_I$ violates an inequality (3.47).*

Proof. We first prove the correctness of the algorithm, assuming that no clique inequality has been found to be violated by the particular fractional point. Since the number of variables in the inequality is 16 and the r.h.s. is 2, at least one variable must possess a value greater than $\frac{1}{8}$. If this variable, namely x_c, is equal to 1, then 10 other variables having at least 2 indices in common with x_c will be set to 0. The remaining 5 variables, which have 0 or 1 index in common with x_c, belong to a clique of type II, therefore cannot sum up to a value greater than 1. Therefore, variable x_c being 1 implies that inequality (3.48) cannot be violated. This proves the validity of the bounds at *Step I.*

As an example, let $c = (i_0, j_0, k_0, l_0)$ and consider inequality (3.48). The 5 variables not set to 0 if $x_c = 1$ are $\{x_{i_0j_1k_1l_1}, x_{i_1j_0k_1l_1}, x_{i_1j_1k_0l_1}, x_{i_1j_1k_1l_0}, x_{i_1j_1k_1l_1}\}$ and all belong to the node set of the clique of type II $Q(d)$, for $d = (i_1, j_1, k_1, l_1)$.

Since 5 variables belong to a clique of type II and no such inequality is violated, the sum of these 5 variables will be no more than 1. Therefore, the antiweb inequality will be violated only if the sum of remaining 11 variables is greater than 1. Since one of those variables is already greater than $\frac{1}{8}$ (variable x_c of the algorithm), at least one of the remaining 10 must be greater than $\frac{1-x_c}{10}$. This can be a variable having either 2 or 3 indices in common with x_c. The remaining steps attempt to find such a variable.

Concerning the complexity of the algorithm, *Step I* is performed up to n^4 times. *Step II* will be performed at most $8n^2$ times (see Lemma 3.15). For a particular x_c, the variables x_d, such that $|c \cap d| = 2$, are $6(n-1)^2$, as shown in the proof of Theorem 3.19. Therefore, it takes up to $8n^2 \cdot 6(n-1)^2$, i.e. $O(n^4)$, steps to identify all such ordered pairs (c,d). This is the complexity of *Step II.*

Taking into account that a particular c appears at exactly six constraints, the number of $d's$ such that $x_d > \frac{1-x_c}{10}$ cannot be more than 10 in each constraint, or 60 in total. This implies that *Step III* will be performed at most 60× $8n^2$ times. For c, d given, the number of $g's$ such that $|c \cap g| = 0, |d \cap g| = 2$ is exactly $(n-1)^2$. Checking whether the inequality $X(A(c,g)) \leq 2$ is violated involves a constant number of steps. Overall, the complexity of *Step III* is $60 \cdot 8n^2 \cdot (n-1)^2$, i.e.$O(n^4)$.

Similarly, *Step IV* is implemented up to $8n^2 \cdot 4(n-1)$ times in order to identify all ordered pairs (c,d), therefore the complexity of this step is $O(n^3)$. Following the same reasoning as before, the number of $d's$ such that $|c \cap d| = 3$, $x_d > \frac{1-x_c}{10}$, cannot be more that 60. For c, d fixed, finding all $f's$ of *Step V* involves $3(n-1)$ steps. As a result, *Step V* will be performed $8n^2 \cdot 60 \cdot 3(n-1)$, i.e. $O(n^3)$ times.

Finally, the number of $f's$ can be up to 54. For a particular pair of (c, f), the number of $g's$ examined at *Step VI* are exactly $(n-1)^2$. Overall, *Step VI* is implemented $8n^2 \cdot 60 \cdot 54 \cdot (n-1)^2$, i.e. $O(n^4)$. The complexity of each step of the algorithm being at most $O(n^4)$ implies that the algorithm is also of linear complexity with respect to the number of variables. ■

3.6 Composite clique inequalities

Composite clique inequalities were introduced in Section 2.6. This section proves that they induce facets of P_I. The proof is based on condition *(d)* of Theorem 3.1, exactly as the proofs in [2]. Let us present again the form of the inequality, for $c_n = (n,n,n,n), c_0 = (i_0, j_0, k_0, l_0)$:

$$\begin{aligned} 2x_{nnnn} + \sum_{i \neq n} x_{innn} + \sum_{j \neq n} x_{njnn} + \sum_{k \neq n} x_{nnkn} + \sum_{l \neq n} x_{nnnl} & \\ + x_{i_0j_0nn} + x_{i_0nk_0n} + x_{nj_0k_0n} + x_{i_0nnl_0} + x_{nj_0nl_0} + x_{nnk_0l_0} & \leq 2 \end{aligned} \tag{3.49}$$

Theorem 3.23 *For $n \geq 7$, (3.49) defines a facet of P_I.*

Proof. Since (3.49) is induced by a wheel, its validity for all $x \in P_I$ arises from the intersection graph. Let $P_I(W(c_n, c_0)) = \{x : x \in P_I, \sum\{x_q : q \in W(c_n, c_0)\} = 2\}$. Denote as $\mathring{x}$ a point of P_I corresponding to a standardised pair of OLS, i.e. $\mathring{x}_{t1tt} = 1, \forall t \in \{1, \dots, n\}$

Let $x = \mathring{x}(1 \leftrightarrow n)_2$. Point $x \in P_I(W(c_n, c_0))$, since $x_{nnnn} = 1$; hence, $P_I(W(c_n, c_0)) \neq \emptyset$.

To show that $P_I \setminus P_I(W(c_n, c_0)) \neq \emptyset$, we consider again the point $\mathring{x}$ We consider two cases, viz $\mathring{x}_{nnk_0l_0} = 1, \mathring{x}_{nnk_0l_0} \neq 1$.

Case (i): If $x_{nnk_0l_0} = 1$, there exists $j_1 \in J \setminus \{1, n\}$ such that $(k(i_0, j_1), l(i_0, j_1)) \neq (k_0, n), (n, l_0)$. Let $x = \mathring{x}(j_1 \leftrightarrow n)_2$. Clearly, $x \in P_I \setminus P_I(W(c_n, c_0))$.

Case (ii): For $x_{nnk_0l_0} \neq 1$, if $\mathring{x}_{i_0nk_0n} = 0$ and $\mathring{x}_{i_0nnl_0} = 0$ it follows that $\mathring{x} \in P_I \setminus P_I(W(c_n, c_0))$. Otherwise, assume w.l.o.g. that $\mathring{x}_{i_0nk_0n} = 1$. Observe that there must exist a column $j_1 \in J \setminus \{1, n\}$, such that $(k(i_0, j_1), l(i_0, j_1)) \neq (n, l_0)$ and $(k(n, j_1), l(n, j_1)) \neq (k_0, l_0)$. Let $x = \mathring{x}(j_1 \leftrightarrow n)_2$. Clearly, $x \in P_I \setminus P_I(W(c_n, c_0))$.

Let $a \in \mathbb{R}^{n^4}$, $a_0 \in \mathbb{R}$ such that $ax = a_0$, for every $x \in P_I(W(c_n, c_0))$. The proof established that (a, a_0) is bound to be a linear combination of the rows of A and inequality (2.26). This implies that there exist scalars λ^1_{kl}, λ^2_{il}, λ^3_{jl}, λ^4_{ij}, λ^5_{jk}, λ^6_{ik}, $\pi \in \mathbb{R}$, $\forall i \in I, j \in J, k \in K, l \in L$, such that:

$$a_{ijkl} = \begin{cases} \lambda^1_{kl} + \lambda^2_{il} + \lambda^3_{jl} + \lambda^4_{ij} + \lambda^5_{jk} + \lambda^6_{ik}, & (i,j,k,l) \in C \setminus W(c_n, c_0) \\ \lambda^1_{kl} + \lambda^2_{il} + \lambda^3_{jl} + \lambda^4_{ij} + \lambda^5_{jk} + \lambda^6_{ik} + \pi, & (i,j,k,l) \in W(c_n, c_0) \setminus \{c_n\} \\ \lambda^1_{nn} + \lambda^2_{nn} + \lambda^3_{nn} + \lambda^4_{nn} + \lambda^5_{nn} + \lambda^6_{nn} + 2\pi & (i,j,k,l) = (n,n,n,n) \end{cases} \tag{3.50}$$

$$a_0 = \sum_{k \in K, l \in L} \lambda^1_{kl} + \sum_{i \in I, l \in L} \lambda^2_{il} + \sum_{j \in J, l \in L} \lambda^3_{jl} + \sum_{i \in I, j \in J} \lambda^4_{ij} + \sum_{j \in J, k \in K} \lambda^5_{jk} + \sum_{i \in I, k \in K} \lambda^6_{ik} + 2\pi \tag{3.51}$$

Define the scalars $\lambda^1_{kl}, \dots, \lambda^6_{ik}$ as in Theorem 3.5.

To study the case of $(i,j,k,l) \in C \setminus W(c_n, c_0)$, we need a number of intermediate results.

Lemma 3.24 *For $n \geq 7$, $i_q, i_1 \in I \setminus \{1, n\}$, $j_q, j_1 \in J \setminus \{1, n\}$, $k_2, k_3 \in K \setminus \{1, k_0, n\}$ where $k_0 \in K \setminus \{1, n\}$, $l_1 \in L \setminus \{1, l_0, n\}$, $l_2 \in L \setminus \{1, l_0\}, l_3 \in L \setminus \{1\}$ where $l_0 \in L \setminus \{1, n\}$, there exists $x \in P_I$ as illustrated in Table 3.14*

Proof. At point $\mathring{x}$, let $i_1 \in I \setminus \{1, n\}$, $j_1 \in J \setminus \{1\}$ be such that $k(i_1, j_1) = 1$, $l(i_1, j_1) = n$. W.l.o.g. assume that there exists $i_q \in I \setminus \{1, i_1, n\}$, such that $k(i_q, 1) = k_1$. Let also $i_2 \in I \setminus \{i_1, n\}$, $j_2 \in I \setminus \{j_1, n\}$ be such that $\mathring{x}_{i_2j_2k_1n} = 1$. By denoting i_3, j_3 the row and column at which the pair (k_2, n) appears ($\mathring{x}_{i_3j_3k_2n} = 1, i_3 \in I \backslash \{i_1, i_2, n\}, j_3 \in I \backslash \{j_1, j_2, n\}$) and by performing the interchanges

Table 3.14: Point x (Lemma 3.24)

	1	$\cdots$	j_q	$\cdots$	j_1	$\cdots$	n
1	11		k_2l_2				
$\vdots$							
i_q	nl_1		k_3l_3				
$\vdots$							
i_1					$1n$		
$\vdots$							
n							nn

$(k_2 \leftrightarrow k_1)_3(i_3 \leftrightarrow i_2)_1(j_3 \leftrightarrow j_2)_2$, we derive point $\mathring{x}$ with $i_q \neq i_1$. In any case, if we denote $l(i_q, 1)$ as l_1, we have $\mathring{x}_{i_q 1 k_1 l_1} = \mathring{x}_{i_1 j_1 1 n} = \mathring{x}_{i_2 j_2 k_1 n} 1$. Let $\tilde{x} = \mathring{x}(i_2 \leftrightarrow n)_1(j_2 \leftrightarrow n)_2$. In Table 3.15, it is shown that for $n \geq 7$, at point $\tilde{x}$, there exists $j_q \in J \setminus \{1, n\}$ such that $k(i_q, j_q), k(1, j_q) \in K \setminus \{1, n\}$, $l(i_q, j_q), l(1, j_q) \in K \setminus \{1, n\}$

Table 3.15: Point $\tilde{x}$ (Lemma 3.24)

	1	$\cdots$	j_q	$\cdots$	j_1	$\cdots$	j_3	$\cdots$	j_4	$\cdots$	j_5	$\cdots$	n
1	11		k_2l_2								$n\cdot$		
$\vdots$													
i_q	nl_1		k_3l_3				$1\cdot$		$\cdot 1$				
$\vdots$													
i_1					$1n$								
$\vdots$													
n													nn

Observe that we can safely assume $k(i_q, j_q), k(1, j_q) \neq k_0$. To prove this, consider $k(i_q, j_q) = k_0$. Then there exists $k_t \in K \setminus \{1, k_0, n\}$ such that the interchange $(k_t \leftrightarrow k_0)$ will set $k(i_q, j_q) = k_t \neq k_0$. By the same argument, $l(i_q, j_q), l(1, j_q), l_1 \neq l_0$ If we denote $k(1, j_q), k(i_q, j_q)$ as k_2, k_3 and $l(1, j_q), l(i_q, j_q)$ as l_2, l_3, we obtain point x of Table 3.14. ■

(Back to the proof of theorem 3.23) For $(i, j, k, l) \in C \setminus W(c_n, c_0)$, if we substitute λs in (3.50), we obtain

$$\begin{aligned} a_{ijkl} &= a_{ij11} + a_{i1k1} + a_{i11l} + a_{1jk1} + a_{1j1l} + a_{11kl} \\ &\quad -2a_{i111} - 2a_{1j11} - 2a_{11k1} - 2a_{111l} + 3a_{1111} \end{aligned} \tag{3.52}$$

Observe that (3.52) is true, if at least two of the indices are equal to one. All the other cases are defined with respect to the number of indices equal to n. In other words, for $(i, j, k, l) \in C \setminus W(c_n, c_0)$, we will prove the validity of eq. (3.50) where none, one, or two of the indices are equal to n. Within each case, certain subcases are proved, depending on the number of indices allowed to take value 1. For the first two cases, eq. (3.50) is shown via proving eq. (3.52), whereas for the last case eq. (3.50) is shown explicitly.

Another critical note is that all cases will be carried out by applying Proposition 3.6. Each time the equation of this proposition is used, the existence of 4 distinct vectors of P_I is implied. Hence, we are incorporating collections of points rather than single points. The correctness of this proof requires that all these points belong also to the face $P_I(W(c_n, c_0))$.

Case 1: Let $(i,j,k,l) = (i_q, j_q, k_q, l_q)$, where $i_q \neq n, j_q \neq n, k_q \neq n, l_q \neq n$.

Let also x^1 denote point x of Lemma 3.24. For $n \geq 5$, there exists $k_q \in K \setminus \{1, k_2, k_3, n\}$ such that $x^2 = x^1(1 \leftrightarrow k_q)_3$. For $n \geq 6$, there exists $l_q \in L \setminus \{1, l_1, l_2, l_3, n\}$ such that $x^3 = x^2(1 \leftrightarrow l_q)_4$, $x^4 = x^1(1 \leftrightarrow l_q)_4$. Let X_1, X_2, X_3, X_4 denote the collections of points for each variable. Notice that $X_1, X_2, X_3, X_4 \in P_I(W(c_n, c_0))$, since at all points of these collections have $(k(n,n), l(n,n)) = (n,n)$.

Subtracting equation $x^2((1,i_q)_1;(1,j_q)_2)$ from $x^1((1,i_q)_1;(1,j_q)_2)$ yields eq. (3.52) for the coefficient $a_{i_q j_q k_q 1}$.The correctness of eq. (3.52) for $a_{1 j_q k_q l_q}, a_{i_q 1 k_q l_q}, a_{i_q j_q 1 l_q}$ follows by symmetry.

Similarly, subtracting equation $x^4((1,i_q)_1;(1,j_q)_2)$ from $x^3((1,i_q)_1;(1,j_q)_2)$ results in

$$a_{i_q j_q k_q l_q} = (a_{i_q j_q 1 l_q} + a_{i_q 1 k_q l_q} + a_{1 j_q k_q l_q}) - a_{i_q 1 1 l_q} - a_{1 j_q 1 l_q} - a_{1 1 k_q l_q} + a_{1 1 1 l_q}$$

Substituting terms in brackets from eq. (3.52) for $a_{1 j_q k_q l_q}, a_{i_q 1 k_q l_q}, a_{i_q j_q 1 l_q}$, we obtain, after arithmetic operations, eq. (3.52) for $a_{i_q j_q k_q l_q}$.

Case 2: Let $(i,j,k,l) = (i_q, j_q, n, l_q)$, where $i_q \neq n, j_q \neq n, l_q \neq n$.

As in the previous case, at point x of Lemma 3.24, let $k_q \in K \setminus \{1, k_2, k_3, n\}$ and $l_q \in L \setminus \{1, l_1, l_2, l_3, n\}$. Derive points $x^1 = x(l_1 \leftrightarrow l_q)_4$ and $x^2 = x^1(1 \leftrightarrow n)_3(i_1 \leftrightarrow n)_1(j_1 \leftrightarrow n)_2$. Again, $X_1, X_2 \in P_I(W(c_n, c_0))$, since at all points of these collections $(k(n,n), l(n,n)) = (n,n)$. Subtracting equation $x^2((1,i_q)_1;(1,j_q)_2)$ from $x^1((1,i_q)_1;(1,j_q)_2)$ results in:

$$\begin{aligned} & a_{1111} + a_{1 j_q n 1} + a_{i_q 1 n 1} + a_{i_q j_q 1 1} - (a_{i_q j_q n 1} + a_{i_q 1 1 1} + a_{1 j_q 1 1} + a_{1 1 n 1}) \\ = \quad & a_{i_q j_q n l_q} + a_{i_q 1 1 l_q} + a_{1 j_q 1 l_q} + a_{1 1 n l_q} - (a_{i_q j_q 1 l_q} + a_{i_q 1 n l_q} + a_{1 j_q n l_q} + a_{1 1 1 l_q}) \quad (3.53) \end{aligned}$$

Let $\dot{x}^1 = x^1(l_1 \leftrightarrow l_q)_4$ and $\dot{x}^2 = x^2(l_1 \leftrightarrow l_q)_4$. Notice $\dot{X}_1, \dot{X}_2 \in P_I(W(c_n, c_0))$, since $(k(n,n), l(n,n)) = (n,n)$ at all points of these collections. Adding the difference $\dot{x}^1((1,i_q)_1;(1,j_q)_2) - \dot{x}^2((1,i_q)_1;(1,j_q)_2)$ to (3.53) yields:

$$\begin{aligned} & 2(a_{1111} + a_{1 j_q n 1} + a_{i_q 1 n 1} + a_{i_q j_q 1 1} - (a_{i_q j_q n 1} + a_{i_q 1 1 1} + a_{1 j_q 1 1} + a_{1 1 n 1})) \\ = \quad & \sum_{l \in \{l_1, l_q\}} \{a_{i_q j_q n l} + a_{i_q 1 1 l} + a_{1 j_q 1 l} + a_{1 1 n l} - (a_{i_q j_q 1 l} + a_{i_q 1 n l} + a_{1 j_q n l} + a_{1 1 1 l})\} \quad (3.54) \end{aligned}$$

Let $x^3 = x^2(1 \leftrightarrow l_q)_4(1 \leftrightarrow l_1)_4$, $x^4 = x(1 \leftrightarrow l_q)_4$. Again $X_3, X_4 \in P_I(W(c_n, c_0))$. The difference $x^3((1,i_q)_1;(1,j_q)_2) - x^4((1,i_q)_1;(1,j_q)_2)$ shows that the right-hand side of (3.54) is 0, therefore proving (3.52) for the coefficient $a_{i_q j_q n 1}$. The correctness of eq. (3.52) for coefficients $a_{i_q 1 n l_q}, a_{1 j_q n l_q}$ follow by symmetry.

Because of (3.52), taken for $a_{i_q j_q n 1}$, the right-hand side of (3.53) is equal to 0, hence:

$$a_{i_q j_q n l_q} + a_{i_q 1 1 l_q} + a_{1 j_q 1 l_q} + a_{1 1 n l_q} - (a_{i_q j_q 1 l_q} + a_{i_q 1 n l_q} + a_{1 j_q n l_q}) - a_{1 1 1 l_q}$$

Substituting terms in brackets from (3.52)for $a_{i_q j_q 1 l_q}, a_{i_q 1 n l_q}, a_{1 j_q n l_q}$, we obtain (3.52) for $a_{i_q j_q n l_q}$. By reversing the roles of the sets, and following the same procedure, we can derive eq. (3.52)for $a_{i_q j_q n l_q}, a_{i_q j_q k_q n}, a_{i_q n k_q l_q}$.

Case 3: Let $(i,j,k,l) = (n, n, k_q, l_q)$, where $k_q \neq n, l_q \neq l_0, n$.

In this case, we prove (3.50) directly. First, we need to prove the following lemma.

Lemma 3.25 *For $n \geq 4$ and $k_1 \in K \setminus \{1, k_0, n\}$, the point x illustrated in the Table 3.16, exists in P_I.*

Table 3.16: Point x (Lemma 3.25)

	$\cdots$	j_0	$\cdots$	n
$\vdots$				
i_0				nl_0
$\vdots$				
i_1				k_1n
$\vdots$				
n		nn		11

Proof. Let $\tilde{x} = \mathring{x}(l(1,n) \neq l_0 ? l(1,n) \leftrightarrow l_0)_4(1 \leftrightarrow n)_2(1 \leftrightarrow n)_1(n \leftrightarrow l_0)_4$. Therefore, $x_{i_1 n k_{(i_1,n)} l_{(i_1,n)}} = 1$. If $k(i_1,n) \neq k_0$ then we denote it as k_1 and derive point x as illustrated in Table 3.16. Otherwise, for $n \geq 4$, there exists $k_1 \in K \setminus \{1, k_0, n\}$ such that $x = \tilde{x}(k_1 \leftrightarrow k_0)_3$. ■

(Back to Case 3) Let x^1 denote point x of Lemma 3.25. There exists $k_q \in K \setminus \{1, k_1, n\}$, therefore let $x^2 = x^1(1 \leftrightarrow k_q)_3$. Observe that $x^1, x^2 \in P_I(W(c_n, c_0))$, since at both points we have $(k(n, j_0), l(n, j_0)) = (n, n)$ and $(k(i_0, n), l(i_0, n)) = (n, l_0)$. The equation $ax = a_0$ must be satisfied by both x^1 and x^2, hence $ax^1 = ax^2$ or

$$\begin{aligned} & a_{nn11} + \sum_{i \in I \setminus \{n\}} a_{ij(i,1)1l(i,1)} + \sum_{i \in I} a_{ij(i,k_q)k_q l(i,k_q)} \\ = \; & a_{nnk_q1} + \sum_{i \in I \setminus \{n\}} a_{ij(i,1)k_q l(i,1)} + \sum_{i \in I} a_{ij(i,k_q)1l(i,k_q)} \end{aligned} \tag{3.55}$$

All terms in eq. (3.55), except a_{nnk_q1}, a_{nn11}, have at most one index with the value n, which implies that for each of them eq. (3.50) has shown to be true in one of the previous cases. If we solve eq. (3.55) with respect to term a_{nnk_q1}, substitute terms in the summands from eq. (3.50), and perform certain arithmetic, we can derive the following expression:

$$\begin{aligned} a_{nnk_q1} = \; & a_{nn11} \\ & + \sum_{i \in I} \{\lambda^1_{1l(i,1)} + \lambda^1_{k_q l(i,k_q)} - \lambda^1_{k_q l(i,1)} - \lambda^1_{1l(i,k_q)}\} \\ & + \sum_{i \in I} \{\lambda^5_{j(i,1)1} + \lambda^5_{j(i,k_q)k_q} - \lambda^5_{j(i,1)k_q} + \lambda^5_{j(i,k_q)1}\} \\ & + \lambda^1_{k_q l(n,1)} + \lambda^5_{j(n,1)k_q} + \lambda^6_{nk_q} \\ & - \lambda^1_{1l(n,1)} - \lambda^5_{j(n,1)1} - \lambda^6_{n1} \end{aligned} \tag{3.56}$$

The critical observation is that:

$$\sum_{i\in I}\lambda^1_{1l(i,1)} = \sum_{i\in I}\lambda^1_{1l(i,k_q)}$$

$$\sum_{i\in I}\lambda^1_{k_q l(i,k_q)} = \sum_{i\in I}\lambda^1_{k_q l(i,1)}$$

$$\sum_{i\in I}\lambda^5_{j(i,1)1} = \sum_{i\in I}\lambda^5_{j(i,k_q)1}$$

$$\sum_{i\in I}\lambda^5_{j(i,1)k_q} = \sum_{i\in I}\lambda^5_{j(i,k_q)k_q}$$

Also observe that we can substitute term a_{nn11} from (3.50), because it has two indices equal to one (3.52). Eq. (3.56) becomes:

$$\begin{aligned} a_{nnk_q1} &= \lambda^1_{11}+\lambda^2_{n1}+\lambda^3_{n1}+\lambda^4_{nn}+\lambda^5_{n1}+\lambda^6_{n1} \\ &\quad +\lambda^1_{k_q l(n,1)}+\lambda^5_{j(n,1)k_q}+\lambda^6_{nk_q} \\ &\quad -\lambda^1_{1l(n,1)}-\lambda^5_{j(n,1)1}-\lambda^6_{n1} \end{aligned}$$

Since $l(n,1)=1, j(n,1)=n$ (Table 3.16), the above equation becomes:

$$a_{nnk_q1} = \lambda^1_{k_q1}+\lambda^2_{n1}+\lambda^3_{n1}+\lambda^4_{nn}+\lambda^5_{nk_q}+\lambda^6_{nk_q}$$

which is identical to (3.50) for term a_{nnk_q1}.

Let $x^3 = x^2(1 \leftrightarrow l_q)_4$ where $l_q \in L\setminus\{1,l_0,l_1,n\}$, $x^3 \in P_I(W(c_n,c_0))$. From $ax^2 = ax^3$ we derive

$$\begin{aligned} & a_{nnk_ql_q} + \sum_{i\in I\setminus\{n\}} a_{ij(i,1)k(i,l_q)l_q} + \sum_{i\in I} a_{ij(i,k_q)k(i,l_q)1} \\ = \;& a_{nnk_q1} + \sum_{i\in I\setminus\{n\}} a_{ij(i,1)k(i,1)1} + \sum_{i\in I} a_{ij(i,k_q)k(i,1)1} \end{aligned} \tag{3.57}$$

where the value of the k index, for given i,l is denoted as $k(i,l)$. If we apply to eq. (3.57) transformations similar to those performed on eq. (3.55), we obtain eq. (3.50) for $a_{nnk_ql_q}$. By reversing the roles of the sets, we can illustrate the correctness of eq. (3.50) for all other cases where two of the indices of $(i,j,k,l)\in C\setminus W(c_n,c_0)$ are equal to n.

Our proof concerning all $(i,j,k,l)\in C\setminus W(c_n,c_0)$ is now complete.

For $(i,j,k,l)\in W(c_n,c_0)\setminus\{(n,n,n,n)\}$ define:

$$\pi_{ijkl} = a_{ijkl} - (\lambda^1_{kl}+\lambda^2_{il}+\lambda^3_{jl}+\lambda^4_{ij}+\lambda^5_{jk}+\lambda^6_{ik}) \tag{3.58}$$

To show (3.50), for $(i,j,k,l)\in W(c_n,c_0)\setminus\{(n,n,n,n)\}$, it is sufficient to prove that all π_{ijkl} are equal. We proceed in a series of steps.

Step1: For (i_1,j_1,k_1,l_1), $(i_2,j_2,k_2,l_2)\in W(c_n,c_0)$ such that $|(i_1,j_1,k_1,l_1)\cap c_n| = 2$, $|(i_1,j_1,k_1,l_1)\cap c_0| = 2$ and $|(i_2,j_2,k_2,l_2)\cap c_n| = 2$, $|(i_2,j_2,k_2,l_2)\cap c_0| = 2$, we prove that $\pi_{i_1j_1k_1l_1} = \pi_{i_2j_2k_2l_2} = \pi$.

W.l.o.g. let $(i_1, j_1, k_1, l_1) = (n, j_0, n, l_0), (i_2, j_2, k_2, l_2) = (n, j_0, k_0, n)$. Consider the point $\hat{x}$ illustrated in Table 3.17

Table 3.17: Point $\hat{x}$ (Step 1)

	$\cdots$	j_0	$\cdots$	j_1	$\cdots$	n
$\vdots$						
i_0						nn
$\vdots$						
n		nl_0		k_0n		

It is not difficult to establish the existence of this point: at point $\mathring{x}$ let $i_1 \in I \backslash \{1, n\}$ be such that $k(i_1, 1) = k_0$, $l(i_1, 1) = l_1$. Derive point $x^1 = \mathring{x}(l_1 \leftrightarrow n)_4$ $(l_1 \neq l_0 ? l_1 \leftrightarrow l_0)_4$ $(1 \leftrightarrow n)_2(n \leftrightarrow i_0)_1$. Observe that pair $(n, n) \in K \times L$ does not appear in rows i_0, i_1 and column n. Thus, we can place it in column j_0 row n, by performing the necessary row and/or column interchanges, without affecting the positions of pairs nl_0 (row i_0, column n) and pairs k_0n (row i_1, column n). Then, by interchanging the roles of the sets I, J with respect to the rows and columns of the OLS structure, we finally derive point $\hat{x}$.

Let $\tilde{x} = \hat{x}(j_1 \leftrightarrow j_0)_2$. Observe that $\hat{x} \in P_I(W(c_n, c_0))$ $(\hat{x}_{i_0nnn} = \hat{x}_{nj_0nl_0} = 1)$, and $\tilde{x} \in P_I(W(c_n, c_0))$ $(\tilde{x}_{i_0nnn} = \tilde{x}_{nj_0k_0n} = 1)$. Hence, equation $a\hat{x} = a\tilde{x}$ implies

$$\begin{aligned} & a_{nj_0nl_0} + \sum_{i \neq n} a_{ij_0k(i,j_0)l(i,j_0)} + a_{nj_1k_0n} + \sum_{i \neq n} a_{ij_1k(i,j_1)l(i,j_1)} \\ = \quad & a_{nj_0k_0n} + \sum_{i \neq n} a_{ij_0k(i,j_1)l(i,j_1)} + a_{nj_1nl_0} + \sum_{i \neq n} a_{ij_1k(i,j_0)l(i,j_0)} \end{aligned}$$

If we substitute the first terms of both sides from (3.58), the rest of the terms from (3.50) and cancel out identical terms, we obtain $\pi_{nj_0k_0n} = \pi_{nj_0nl_0} = \pi$.

By reversing the roles of the sets and applying similar procedures, we obtain

$$\pi_{i_0j_0nn} = \pi_{i_0nk_0n} = \pi_{nj_0k_0n} = \pi_{i_0nnl_0} = \pi_{nj_0nl_0} = \pi_{nnk_0l_0} = \pi$$

Step 2: For (i_1, j_1, k_1, l_1), $(i_2, j_2, k_2, l_2) \in W(c_n, c_0)$ such that $((i_1, j_1, k_1, l_1) \cap c_n) = ((i_2, j_2, k_2, l_2) \cap c_n)$ and $|(i_1, j_1, k_1, l_1) \cap c_n| = |(i_2, j_2, k_2, l_2) \cap c_n| = 3$, we will show $\pi_{i_1j_1k_1l_1} = \pi_{i_2j_2k_2l_2}$.

W.l.o.g. assume that $(i_1, j_1, k_1, l_1) = (n, j_1, n, n)$, $(i_2, j_2, k_2, l_2) = (n, j_2, n, n)$, with $j_1, j_2 \in J \setminus \{n\}$. Consider the point $\hat{x}$ illustrated in Table 3.18.

Table 3.18: Point $\hat{x}$ (Step 2)

	$\cdots$	j_1	$\cdots$	j_2	$\cdots$	n
$\vdots$						
i_0						
$\vdots$						
n		nn		k_1l_1		k_0l_0

Again, it is easy to establish existence for this point: at point $\mathring{x}$, let $i_1 \in I \setminus \{1, n\}$ be such that $k(i_1, 1) = k_0$. Denote $l(i_1, 1)$ as l_2. Then, $x^1 = \mathring{x}(l_2 \neq l_0 ? l_2 \leftrightarrow l_0)_4(i_1 \leftrightarrow n)_1$. Let

$i_2 \in I \setminus \{1, n\}$ be such that $k(i_2, 1) = k_1$. We denote $l(i_2, 1)$ as l_1 and perform the interchange $(1 \leftrightarrow n)_2$. At the derived point we reverse the roles of the sets I, J, thus obtaining point $\hat{x}$ of Table 3.18. Let $\tilde{x} = \hat{x}(j_1 \leftrightarrow j_2)_2$. Observe that $\hat{x} \in P_I(W(c_n, c_0))$ $(\hat{x}_{nj_1nn} = \hat{x}_{nnk_0l_0} = 1)$ and $\tilde{x} \in P_I(W(c_n, c_0))$ $(\tilde{x}_{nj_2nn} = \tilde{x}_{nnk_0l_0} = 1)$. Thus, $a\hat{x} = a\tilde{x}$ implies

$$\begin{aligned} & a_{nj_1nn} + \sum_{i \neq n} a_{ij_1k(i,j_1)l(i,j_1)} + a_{nj_2k_1l_1} + \sum_{i \neq n} a_{ij_2k(i,j_2)l(i,j_2)} \\ = \quad & a_{nj_2nn} + \sum_{i \neq n} a_{ij_2k(i,j_1)l(i,j_1)} + a_{nj_1k_1l_1} + \sum_{i \neq n} a_{ij_1k(i,j_2)l(i,j_2)} \end{aligned}$$

If we substitute the first terms of both sides from (3.58), the rest of the terms from (3.50) and cancel out identical terms, we obtain $\pi_{nj_1nn} = \pi_{nj_2nn} = \pi^2, \forall j_1, j_2 \in J \setminus \{n\}$.

Reversing the roles of the sets is enough to prove:

$$\begin{aligned} \pi_{i_1nnn} &= \pi_{i_2nnn} = \pi^1, \forall i_1, i_2 \in I \setminus \{n\} \\ \pi_{nnk_1n} &= \pi_{nnk_2n} = \pi^3, \forall k_1, k_2 \in K \setminus \{n\} \\ \pi_{nnnl_1} &= \pi_{nnnl_2} = \pi^4, \forall l_1, l_2 \in L \setminus \{n\} \end{aligned}$$

Step 3: We will show that $\pi^1 = \pi^2 = \pi^3 = \pi^4$.

Let $\breve{x} = \hat{x}(j_1 \leftrightarrow n)_2$ where $\hat{x}$ is the point illustrated in Table 3.17 Let $\tilde{x} = \breve{x}(i_0 \leftrightarrow n)_1$. Note that $\breve{x} \in P_I(W(c_n, c_0))$ $(\breve{x}_{nj_0nl_0} = \breve{x}_{nnk_0n} = 1)$ and $\tilde{x} \in P_I(W(c_n, c_0))$ $(\tilde{x}_{nj_1nn} = \tilde{x}_{i_0nk_0n} = 1)$. Thus, $a\breve{x} = a\tilde{x}$ implies

$$\begin{aligned} & a_{nnk_0n} + a_{nj_0nl_0} + \sum_{j \neq j_0, n} a_{njk(n,j)l(n,j)} + a_{i_0j_1nn} + \sum_{j \neq j_1} a_{i_0jk(i_0,j)l(i_0,j)} \\ = \quad & a_{nj_1nn} + a_{i_0nk_0n} + \sum_{j \neq j_1} a_{njk(i_0,j)l(i_0,j)} + a_{i_0j_0nl_0} + \sum_{j \neq j_0, n} a_{i_0jk(n,j)l(n,j)} \end{aligned}$$

Substituting the first two terms of both sides from eq. (3.58), the rest of the terms from eq. (3.50) and taking into account the results obtained in the previous two steps we obtain $\pi_{nnk_0n} = \pi_{nj_1nn} \Rightarrow \pi^3 = \pi^2$. As in the previous steps, by reversing the roles of the sets and following the same procedure, we obtain

$$\pi^1 = \pi^2 = \pi^3 = \pi^4 = \kappa$$

Step 4: We will prove that $\kappa = \pi$.

Consider the point $\hat{x}$ illustrated in Table 3.19.

It is easy to see that this point exists. At an arbitrary point $x \in P_I$ pairs $(nl_0), (nn)$ cannot lie at the same row or column. Thus we can easily derive a point $\breve{x}$ such that $\breve{x}_{nj_0nl_0} = \breve{x}_{i_1nnn} = 1$. Also let $\breve{x}_{i_pj_pk_0n} = 1$. Clearly, $j_p \neq n, i_p \neq i_1$. If $i_p \neq n$ and $j_p \neq j_0$ then $\hat{x} = \breve{x}(i_p \neq i_0 ? i_p \leftrightarrow i_0)_1 (j_p \neq j_1 ? j_p \leftrightarrow j_1)_2$. Otherwise assume w.l.o.g. that $i_p = n$. There exist $i_2 \in I \setminus \{i_1, n\}$, $j_2 \in J \setminus \{j_0, j_p, n\}$, $k_1 \in K \setminus \{k_0, n\}$ such that $\breve{x}_{i_2j_2k_1n} = 1$. Derive point $\hat{x} = \breve{x}(i_2 \neq i_0 ? i_2 \leftrightarrow i_0)_1 (j_2 \neq j_1 ? j_2 \leftrightarrow j_1)_2 (k_1 \leftrightarrow k_0)_3$.

Now, let $\tilde{x} = \hat{x}(j_1 \leftrightarrow n)_2$. Note that $\tilde{x} \in P_I(W(c_n, c_0))$ $(\tilde{x}_{nj_0nl_0} = \tilde{x}_{i_1nnn} = 1)$ and $\tilde{x} \in$

Table 3.19: Point $\hat{x}$ (Step 4)

	$\cdots$	j_0	$\cdots$	j_1	$\cdots$	n
$\vdots$						
i_0				k_0n		
$\vdots$						
i_1						nn
$\vdots$						
n		nl_0				

$P_I(W(c_n, c_0))$ $(\tilde{x}_{nj_0nl_0} = \tilde{x}_{i_0nk_0n} = 1)$. Equation $a\hat{x} = a\tilde{x}$ yields:

$$\begin{aligned} & a_{i_1nnn} + \sum_{i \neq i_1} a_{ink(i,n)l(i,n)} + a_{i_0j_1k_0n} + \sum_{i \neq i_0} a_{ij_1k(i,j_1)l(i,j_1)} \\ = \; & a_{i_0nk_0n} + \sum_{i \neq i_1} a_{ij_1k(i,n)l(i,n)} + a_{i_1j_1nn} + \sum_{i \neq i_0} a_{ink(i,j_1)l(i,j_1)} \end{aligned}$$

Substituting the first terms of both sides from (3.58), the rest of the terms from (3.50) and cancelling equivalent terms, we obtain $\pi_{i_1nnn} = \pi_{i_0nk_0n}$ or $\pi^1 = \pi$ or $\kappa = \pi$.

Finally, we define

$$\pi_{nnnn} = a_{nnnn} - (\lambda^1_{nn} + \lambda^2_{nn} + \lambda^3_{nn} + \lambda^4_{nn} + \lambda^5_{nn} + \lambda^6_{nn}) \tag{3.59}$$

Consider the point $\hat{x}$ illustrated in Table 3.20.

Table 3.20: Point $\hat{x}$ table

	$\cdots$	j_0	$\cdots$	n
$\vdots$				
i_0		nl_0		
$\vdots$				
n				nn

Let $\tilde{x} = \hat{x}(j_0 \leftrightarrow n)_2$. Note that $\hat{x} \in P_I(W(c_n, c_0))$ $(\hat{x}_{nnnn} = 1)$ and $\tilde{x} \in P_I(W(c_n, c_0))$ $(\tilde{x}_{nj_0nn} = \tilde{x}_{i_0nnl_0} = 1)$. Equation $a\hat{x} = a\tilde{x}$, after substituting terms from equations (3.50), (3.58), (3.59) and taking into account the results of Steps 1-4, gives the equation $\pi_{nnnn} = \pi_{nj_0nn} + \pi_{i_0nnl_0} = 2\pi$.

Proving eq. (3.51) proceeds exactly as in Theorem 3.5. The proof is complete. ■

No separation algorithm has been devised for inequalities arising from composite cliques.

3.7 Odd cycle inequalities

As discussed in Section 2.7, the two families of odd hole inequalities identified for the OLS problem arise from the $3AAP_n$ and $3PAP_n$ problems and have been named odd holes of type I and II, respectively. The inequalities of $3AAP_n$ giving rise to odd holes of type I induce facets of the polytope associated with $3AAP_n$; however, a similar result has not been accomplished for the OLS polytope, mainly because of the particularities discussed in Section 3.2. Fortunately, the separation algorithm for the odd hole inequalities of $3AAP_n$ ([10]) is easily extendable and is used to provide an $O(n^4)$ separation procedure for the 5-holes of type I.

Odd holes of type II exist in both $3PAP_n$ and $4PAP_n$, and their Chvátal rank is proved to be 1 in both cases. It is also proved that they define facets of the polytope associated with $3PAP_n$, thus providing a new class of facets for the Latin square polytope ([37]). Unfortunately, the proof is not directly extendable to the OLS polytope, mainly because a completability theorem analogous to Ryser's (in [34]) for Latin squares does not exist for OLS. No separation algorithm for these inequalities is known.

3.7.1 Separation algorithms for odd cycles of type I

This section extends the separation algorithm for lifted odd cycles presented in [10]. The notation and the algorithm are essentially reproduced from this work. Recall that for constructing an inequality of this type we select any 3 out of 4 single ground sets and a single value from the fourth set (see Section 2.7.1). We can assume w.l.o.g. sets I, J, K and value $l_1 \in L$. Subsets $Q_I \subset I$, $Q_J \subset J$, $Q_K \subset K$, satisfying $|Q_I| + |Q_J| + |Q_K| = 2p + 1$, $1 \leq |Q_M| \leq p$ for $M = I, J, K$ are necessary to define the set $Q = Q_I \cup Q_J \cup Q_K$, $|Q| = 2p + 1$. The set of variables in the l.h.s. of the inequality is $X(S(Q, l_1)) = \sum\{x_q : q \in S(Q, l_1)\}$, where $S(Q, l_1) = \sum\{s \in C : |s \cap (Q \cup \{l_1\})| \geq 3\}$. The lifted odd cycle inequality has the form

$$X(S(Q, l_1)) \leq p \tag{3.60}$$

Let us first present the separation algorithm for lifted 5- cycle inequalities. Note that $|Q| = 5$, which implies that one of the subsets Q_I, Q_J, Q_K has cardinality 1, whereas the other two have cardinality 2.

Define:

$$x(i, j, K) = \sum_{k' \in K} x_{ijk'l_1} \tag{3.61}$$

$$x(i, J, k) = \sum_{j' \in J} x_{ij'k'l_1} \tag{3.62}$$

$$x(I, j, k) = \sum_{i' \in I} x_{i'jk'l_1} \tag{3.63}$$

and, for $p \in C$,

$$T(p) = \{q \in C : |p \cap q| = 2 \text{ and } p \cap q \in (I \times J) \cup (I \times K) \cup (J \times K)\}$$

$$X(T(p)) = \sum \{x_q : q \in T(p)\}$$

Assume $p, q \in C$, such that $l_1 \subset p, q$ and $|p \cap q| = 2$, i.e. nodes p and q have two indices in common, one of which is l_1. It is easy to see that $Q \cup \{l_1\} = p \cup q$ for some p, q with these properties. Let w.l.o.g. $p = (i_p, j_p, k_p, l_1)$, $q = (i_q, j_q, k_q, l_1)$. The possible cases for the form of the inequality (3.60), for $p = 2$, are:

Case 3.25.1 $(i_p = i_q)$
$X(S(Q, l_1)) = X(T(p)) + X(T(q)) + x(I, j_p, k_q) + x(I, j_q, k_p) - 2 \cdot (x_{i_p j_p k_q} + x_{i_p j_q k_p})$

Case 3.25.2 $(j_p = j_q)$
$X(S(Q, l_1)) = X(T(p)) + X(T(q)) + x(i_p, J, k_q) + x(i_q, J, k_p) - 2 \cdot (x_{i_p j_p k_q} + x_{i_q j_p k_p})$

Case 3.25.3 ($k_p = k_q$)
$X(S(Q,l_1)) = X(T(p)) + X(T(q)) + x(i_p,j_q,K) + x(i_q,j_p,K) - 2\cdot(x_{i_pj_qk_p} + x_{i_qj_pk_p})$

Alternatively, assume $p,q \in C$, such that $l_1 \subset p,q$ and $|p \cap q| = 3$, i.e. nodes p and q have three indices in common, one of which is l_1. In that case, one index is necessary in order for set $p \cup q$ to become an equivalent for the set $Q \cup \{l_1\}$. Let again $p = (i_p,j_p,k_p,l_1)$, $q = (i_q,j_q,k_q,l_1)$.

Define:

$$x(p,q) = \left\{ \begin{array}{ll} x(i_q,J,k_p) + x(i_q,j_p,K) - 2\cdot x_{i_qj_pk_p}, & \text{if } i_p \neq i_q \\ x(I,j_q,k_p) + x(i_p,j_q,K) - 2\cdot x_{i_pj_qk_p}, & \text{if } j_p \neq j_q \\ x(I,j_p,k_q) + x(i_p,J,k_q) - 2\cdot x_{i_pj_pk_q}, & \text{if } k_p \neq k_q \end{array} \right\} \tag{3.64}$$

In the case of $i_p \neq i_q$, the additional index can be selected either from set J or from set K. If the additional index is $j_1 \in J$, the subsets are $Q_I = \{i_p,i_q\}$, $Q_J = \{j_p,j_1\}$, $Q_K = \{k_p\}$. Selecting an index $k_1 \in K$ implies $Q_I = \{i_p,i_q\}$, $Q_J = \{j_p\}$, $Q_K = \{k_p,k_1\}$. In both cases, the existence of an appropriate set Q has been achieved. Notice that there are two options for selecting an index, for each of the three cases of (3.64), i.e. a total of six cases. The form of the inequality (3.60) for $p = 2$ is:

Case 3.25.4 ($i_p \neq i_q$, $j_1 \in J$)
$X(S(Q,l_1)) = X(T(p)) + x(p,q) + x(I,j_1,k_p) + x(i_p,j_1,K) + x(i_q,j_1,K) - 2\cdot(x_{i_pj_1k_q} + x_{i_qj_1k_q})$

Case 3.25.5 ($i_p \neq i_q$, $k_1 \in K$)
$X(S(Q,l_1)) = X(T(p)) + x(p,q) + x(I,j_p,k_1) + x(i_p,J,k_1) + x(i_q,J,k_1) - 2\cdot(x_{i_pj_pk_1} + x_{i_qj_pk_1})$

Case 3.25.6 ($j_p \neq j_q$, $i_1 \in I$)
$X(S(Q,l_1)) = X(T(p)) + x(p,q) + x(i_1,J,k_p) + x(i_1,j_p,K) + x(i_i,j_q,K) - 2\cdot(x_{i_1j_pk_p} + x_{i_1j_qk_p})$

Case 3.25.7 ($j_p \neq j_q$, $k_1 \in K$)
$X(S(Q,l_1)) = X(T(p)) + x(p,q) + +x(I,j_p,k_1) + x(I,j_q,k_1) + x(i_p,J,k_1) - 2\cdot(x_{i_pj_pk_1} + x_{i_pj_qk_1})$

Case 3.25.8 ($k_p \neq k_q$, $i_1 \in I$)
$X(S(Q,l_1)) = X(T(p)) + x(p,q) + x(i_1,J,k_p) + x(i_1,J,k_q) + x(i_1,j_p,K) - 2\cdot(x_{i_1j_pk_p} + x_{i_1j_pk_q})$

Case 3.25.9 ($k_p \neq k_q$, $j_1 \in J$)
$X(S(Q,l_1)) = X(T(p)) + x(p,q) + x(I,j_1,k_p) + x(I,j_1,k_q) + x(i_p,j_1,K) - 2\cdot(x_{i_pj_1k_p} + x_{i_pj_1k_q})$

Given the above notation, the algorithm for separation of lifted 5-cycle inequalities is given below (see also [10]). Note that the algorithm selects first an index from one set and uses values from the other three sets to construct set Q. The procedure is illustrated assuming that the index selected belongs to set L. Define the set $C_{l_1} = \{c \in C : l_1 \subset c\}$.

Algorithm 3.26 (Separation of lifted 5−cycle inequalities) *Let $x \in P_L \backslash P_I$. For all $l_1 \in L$:*

Step I *Set $d_p = 0$ for all $p \in C_{l_1}$*

Step II *Check x_p for all $p \in C_{l_1}$.*

If $x_p \geq \frac{1}{12n}$ then set $d_p = d_p + x_q$ for all $q \in C_{l_1}$ satisfying $|p \cap q| = 3$

Step III *For all $p \in C_{l_1}$, if $d_p \geq \frac{1}{12}$ then set $d_p = X(T(p))$.*

Step IV *For all $i \in I, j \in J, k \in K$, calculate $x(I,j,k), x(i,J,k), x(i,j,K)$.*

Step V *For all $i \in I, j \in J, k \in K$ form and store the sets:*

$$L(i,J,K) = \{p \in C_{l_1} : d_p > \tfrac{1}{3},\ i_p = i\}$$
$$L(I,j,K) = \{p \in C_{l_1} : d_p > \tfrac{1}{3},\ j_p = j\}$$
$$L(I,J,k) = \{p \in C_{l_1} : d_p > \tfrac{1}{3},\ k_p = k\}$$

Step VI *For each $p = (i_p, j_p, k_p, l_1) \in C_{l_1}$, such that $d_p > \frac{1}{2}$, check d_q for all $q \in L(s)$, where*

$$L(s) = (L(i_p,J,K) \cup L(I,j_p,K) \cup L(I,J,k_p))\backslash\{p\}$$

Step VII *If $|p \cap q| = 3$ and $d_q > \frac{2-d_p}{3}$, check whether $X(S(Q,l_1)) > 2$, where $Q = p \cup q$. If yes, stop; otherwise, continue.*

Step VIII *If $|p \cap q| = 2$ and $d_q > \frac{2-d_p}{3}$, check whether $X(S(Q,l_1)) > 2$, for all $Q \supset p \cup q$. If yes, stop; otherwise, continue.*

For a specific l_1,Algorithm 3.26 determines whether an inequality of the form $X(S(Q,l_1)) \leq 2$ is violated in $O(n^3)$ steps. The algorithm has to be repeated for all $4n$ values of all four sets, therefore its total complexity is $O(n^4)$, i.e. linear in the number of variables. Both the correctness and the complexity of the algorithm follow from the results exhibited in [10].

Remark 3.4 *An important fact, arising from results of [10], is that the correctness of Algorithm 3.26 requires that clique inequalities are separated first. In other words, Algorithm 3.26 is guaranteed to provide a violated inequality only if all clique inequalities are satisfied by the given $x \in P_L\backslash P_I$.*

Concerning inequalities arising from odd cycles of larger size, polynomial time separation algorithms can also be devised for particular sub-families. Inequalities (3.60) for $p \geq 3$ need also subsets Q_I, Q_J, Q_K of larger size, therefore the time to detect whether such an inequality is violated is considerably longer. If exactly one of these subsets has cardinality of one, i.e. a single index is selected from exactly one ground set, the resulting family of inequalities is separable in $O(n^{p+1})$ steps for a particular l_1. In total, $O(n^{p+2})$ steps are necessary to detect whether a lifted $2p+1-$cycle inequality of this structure is violated or not. As p increases, the number of steps becomes significantly larger, becoming $O(n^{n+1})$ for $\max\{p\} = n-1$. Because of the excessive computational cost of a separation algorithm, separating inequalities (3.60) for values of p greater than 2 is not considered.

3.7.2 The dimension of odd cycles of type II

This section introduces a new class of facet defining inequalities for the polytope of the $3PAP_n$ problem. Recall from Section 2.7.2 that odd cycles of type II appear in the intersection graph of both the $3PAP_n$ and the $4PAP_n$.

Concerning the $3PAP_n$, the inequality induced by lifting an odd cycle Q of type II has the form:

$$X(Q) = \sum\{x_q : q \in Q\} \leq \left\lfloor \frac{3m}{2} \right\rfloor \tag{3.65}$$

Proposition 3.27 *Inequalities (3.65) are of Chvátal rank 1.*

Proof. *For m being odd, consider the rows of matrix A indexed by $\{i_1j_1,\ i_1k_2,\ j_2k_2,\ldots,\ i_mj_m,\ i_mk_1,\ j_1k_1\}$. By adding up these rows, replacing equality with inequality, dividing the result by 2 and rounding down both sides we derive (3.65). The same can be accomplished for m being even, if we consider the rows of matrix A indexed by $\{i_1j_1,\ i_1k_2,\ j_2k_2,\ldots,\ i_mk_m,\ i_mj_q,\ i_mk_1, j_1k_1\}$. Therefore,* *inequality (3.65) belongs to the elementary closure of (2.12)-(2.14). It is also trivial to construct a fractional solution violating (3.65).* ■

It is easy to see that the corresponding inequalities for the $4PAP_n$ are also of rank at most 1. To prove that they are of rank exactly 1, one must illustrate they define a non-empty face of P_I and also that the induced face does not coincide with P_I. However, no such result has been achieved.

Let us first state a definition and a theorem related to the completability of incomplete Latin squares. Note that an analogous theorem has not been stated for OLS and this is a strong reason why the results of this section are not extendable to the OLS polytope.

Definition 3.1 *Let $r, s \in \{1, ..., n\}$ and L be an incomplete Latin square of order n, whose non-empty cells form an $r \times s$ matrix. Then we call L a Latin rectangle of type (r, s, n).*

Theorem 3.28 (H.J.Ryser) *(in [34]) A Latin rectangle L of type (r, s, n) can be completed to a Latin square of order n if and only if each symbol $i \in \{1, ..., n\}$ occurs at least $r + s - n$ times in L.*

Let us denote as P_{LS} the polytope of $3PAP_n$ or Latin square polytope. The dimension of P_{LS} has been shown to be $(n-1)^3$ (in [37]). Let D denote the matrix of constraints (2.12)-(2.14). Then $P_{LS} = conv\{x \in \{0,1\}^{n^3} : Dx = e\}$, where $e = [1, .., 1]^T \in R^{3n^2}$

Theorem 3.29 *Inequalities (3.65) induce facets if m is odd and $m \leq \frac{n}{2} - 1$ (i.e. $p \leq \frac{3n}{4} - 2$).*

Proof. Assume, w.l.o.g. that the odd cycle is $Q = \{222, 223, 233, 333, ..., (m+1)(m+1)(m+1), (m+1)(m+1)2, (m+1)22\}$. Let $c \in R^{n^3}$ be the $0-1$ incidence vector of the associated inequality:

$$\sum\{x_q : q \in Q\} \leq p \tag{3.66}$$

where $p = \frac{3m-1}{2}$. Let $P_{LS}^{X(Q)} = conv\{x \in \{0,1\}^{n^3} : Dx = e, c^T x = p\}$. We will prove that $\dim(P_{LS}^{X(Q)}) = \dim(P_{LS}) - 1$.

The inequality (3.66) is valid since it is implied by the intersection graph associated with $3PAP_n$. It is also easy to illustrate an $x \in P_{LS}$ such that $c^T x < p$, thus proving that $P_{LS}^{X(Q)} \neq P_{LS}$. Assume finally another inequality $a^T x \leq a_0$, such that $P_{LS}^{X(Q)} \subseteq conv\{x \in \{0,1\}^{n^3} : Dx = e, a^T x = a_0\}$ and consider the scalars (in [37]):

$$\begin{aligned} \lambda^1_{jk} &= a_{1jk} \\ \lambda^2_{ik} &= a_{i1k} - a_{11k} \\ \lambda^3_{ij} &= a_{ij1} - a_{i11} - a_{1j1} + a_{111} \end{aligned} \tag{3.67}$$

By systematically examining all possible cases, we will show that:

$$a_{ijk} = \left\{ \begin{array}{ll} \lambda^1_{jk} + \lambda^2_{ik} + \lambda^3_{ij} + \pi, & (i,j,k) \in Q \\ \lambda^1_{jk} + \lambda^2_{ik} + \lambda^3_{ij}, & (i,j,k) \notin Q \end{array} \right\} \tag{3.68}$$

which is equivalent to proving that, for $\pi > 0$:

$$\begin{aligned} a^T &= \lambda^T D + \pi c^T \\ a_0 &= \lambda^T 1 + \pi \end{aligned} \tag{3.69}$$

This implies that the equality $ax = a_0$ can be expressed as a linear combination of the equality system $\{Dx = e, c^T x = p\}$, which is therefore minimal for $P_I^{X(Q)}$.

It follows from eq. (3.68):

$$a_{222} = \lambda^1_{jk} + \lambda^2_{ik} + \lambda^3_{ij} + \pi$$

Therefore, let

$$\pi = a_{222} - (\lambda^1_{jk} + \lambda^2_{ik} + \lambda^3_{ij})$$

or

$$\pi = a_{222} - a_{122} - a_{212} - a_{221} + a_{211} + a_{121} + a_{112} - a_{111} \tag{3.70}$$

It can be seen by direct substitution that:

$$a_{ijk} = \lambda^1_{jk} + \lambda^2_{ik} + \lambda^3_{ij}$$

if at least one of the indices is equal to 1.

We will first prove (3.68) for $a_{(m+1)22}$. According to (3.68):

$$a_{222} = a_{221} + a_{212} + a_{122} - a_{211} - a_{121} - a_{112} + a_{111} + \pi$$
$$a_{(m+1)22} = a_{(m+1)21} + a_{(m+1)12} + a_{122} - a_{(m+1)11} - a_{121} - a_{112} + a_{111} + \pi$$

Therefore:

$$a_{222} - a_{(m+1)22} = a_{221} - a_{(m+1)21} + a_{212} - a_{(m+1)12} - a_{211} + a_{(m+1)11}$$

or

$$a_{222} + a_{211} + a_{(m+1)12} + a_{(m+1)21} = a_{221} + a_{212} + a_{(m+1)11} + a_{(m+1)22} \tag{3.71}$$

Consider the two Latin rectangles, namely R1 & R2, depicted in Tables 3.21 & 3.22. Note that the emphasised cells correspond to variables whose coefficients appear in (3.71), whereas the bordered cells correspond to variables appearing in the odd cycle inequality. This convention will be kept throughout the rest of this proof. It is also worthwhile to notice the trivial fact that an $r \times s$ rectangle has $r + s - 1$ top-right to bottom-left diagonals.

Table 3.21: Rectangle R1

	1								m+1
2	*1*	*2*	3		4			6	
		3	4	2			6		
	3		2	4	5				
		4		5	6				m-1
						...			
	5		6				m-1	m	
		6					m	m+1	2
m+1	*2*	*1*			m-1			3	m+1

Each rectangle has m rows and $m + 1$ columns, therefore it possesses $2m$ top-right to bottom-left diagonals. The cells of at least $m + 1$ of those diagonals have already been assigned values. The remaining $m - 1$ diagonals can be identically (but not uniquely) completed with elements $m + 2, ..., 2m + 2$ in both rectangles without violating the structure of a Latin square. An example of this construction for $m = 5, n = 6$ is illustrated in Table 3.23.

Table 3.22: Rectangle R2

	1								m+1
2	*2*	*1*	3		4			6	
		3	4	2					
	3		2	4	5				
		4		5	6				
						...			
	5		6				m-1	m	
		6					m	m+1	2
m+1	*1*	*2*						3	m+1

Table 3.23: Rectangle R1 & R2 for m=5

	1					6
2	*1*	*2*	3		4	
		3	4	2		5
	3		2	4	5	
		4		5	6	2
6	*2*	*1*	5		3	6

	1					6
2	*2*	*1*	3		4	
		3	4	2		5
	3		2	4	5	
		4		5	6	2
6	*1*	*2*	5		3	6

Since each rectangle has m rows and $m+1$ columns, the rectangles are completable to Latin squares of order n, if and only if each element appears at least $2m+1-n$ times (see Theorem 3.28). Given that $2m+2 \leq n$, it follows that each element must appear at least 0 times, which is true. Therefore, the two Latin rectangles R1 & R2 can be identically completed to Latin squares L1 & L2 of order n.

Both L1 & L2 define points $x^1, x^2 \in P_I^{X(Q)}$, therefore they must both satisfy equality $a^T x = a_0$, which implies that $a^T x^1 = a^T x^2 = a_0$. By canceling out terms, we derive (3.71).

The proof proceeds in the same fashion for all other cases. We will restrict ourselves to simply illustrating the rectangle R1 for every case. Rectangle R2 is always derived by interchanging the emphasised elements of R1.

One can prove (3.68) for a_{223} by examining the difference $a_{222}-a_{223}$. Given that (3.68) is true for a_{223}, the same can be shown for a_{233} by taking the difference $a_{223}-a_{233}$. Similarly, considering the difference $a_{233}-a_{333}$ proves (3.68) for a_{333}. The remaining coefficients $a_{ijk}, (i,j,k) \in Q$ are examined in the same way. Assuming the ordering $\{a_{222}, a_{223}, a_{233}, a_{333}, \ldots, a_{(m+1)(m+1)(m+1)}, a_{(m+1)(m+1)1}, a_{(m+1)11}\}$, it is not difficult to see that all these comparisons between two consecutive coefficients can be classified into three possible cases.

Case 3.29.1 *We consider the differences $a_{222}-a_{223}, \ldots, a_{(m+1)(m+1)(m+1)}-a_{(m+1)(m+1)2}$. In general, given that (3.68) is true for $a_{ijk}, (i,j,k) \in Q$, we prove that (3.68) is true for $a_{ij(k+1)}, (i,j,k+1) \in Q$ (note that $m+2 \equiv 2$). According to (3.68):*

$a_{ijk} = a_{1jk} + a_{i1k} + a_{ij1} - a_{i11} - a_{1j1} - a_{11k} + a_{111}$

$a_{ij(k+1)} = a_{1j(k+1)} + a_{i1(k+1)} + a_{ij1} - a_{i11} - a_{1j1} - a_{11(k+1)} + a_{111}$

Therefore:

$$a_{ijk} - a_{ij(k+1)} = a_{1jk} - a_{1j(k+1)} + a_{i1k} - a_{i1(k+1)} + a_{11(k+1)} - a_{11k}$$

or

$$a_{11k} + a_{1j(k+1)} + a_{i1(k+1)} + a_{ijk} = a_{11(k+1)} + a_{1jk} + a_{i1k} + a_{ij(k+1)} \tag{3.72}$$

Tables 3.24 and 3.25 illustrate rectangle R1 for k being odd and even, respectively. An example of this construction for m = 5 *is illustrated in Table 3.26.*

Table 3.24: Rectangle R1 for Case 3.29.1, k odd

	1							j				m+1
1	*k*	m+1		3		4		*k+1*				k-1
	m+1	3	k		4						k-1	2
		k	3	4						k-1	2	
	3		4	5						2		
					...			k-1				
	4					k-2	k-1	2				
						k-1	k					
i	*k+1*					2		*k*				
				k-1					...			m
			k-1	2						m-1	m	
		k-1	2							m	m+1	k+1
m+1		2							m		k+1	m+1

Table 3.25: Rectangle R1 for Case 3.29.1, k even

	1								j			m+1
1	*k*			3				5	*k+1*			
		2	3		4		5					
		3	4	2								k-1
	3		2	4	5						k-1	
		4		5	6							
						...						
		5					k-2	k-1				
	5						k-1	k				
i	*k+1*								*k*			
										...		
				k-1							m	
m+1			k-1									2

Table 3.26: Rectangle R1 for m=5 for Case 3.29.1, k odd & even resp.

	1				5	6
1	*5*			3	*6*	4
		3	6		4	2
		6	3	4	2	
	3		4	5		
5	*6*	4	2		*5*	
6	4	2				6

	1			4		6
1	*4*	6		*5*		
	6	2	3			
		3	4			
4	*5*			*4*		
					5	6
6					6	2

Using m elements, we illustrate that m top-right to bottom-left diagonals out of $2m+1$ *can be completed. The rest can be completed using elements* 1, $m+2, \ldots, 2m+2$.*The completion can be performed identically for both* $(m+1, m+1, n)$ *rectangles. The special case of* $k = m+1$ *is treated in exactly the same way, i.e. element* 2 *appears at cell* $(m+1, m+1)$ *of rectangle R1.*

Case 3.29.2 *We consider the differences* $a_{223} - a_{233}, \ldots, a_{(m+1)(m+1)2} - a_{(m+1)22}$. *In general, given that (3.68) is true for* $a_{ijk}, (i,j,k) \in Q$, *we prove that (3.68) is true for* $a_{i(j+1)k}, (i, j+1, k) \in Q$ $(m+2 \equiv 2)$. *According to (3.68):*

$a_{ijk} = a_{1jk} + a_{i1k} + a_{ij1} - a_{i11} - a_{1j1} - a_{11k} + a_{111}$

$a_{i(j+1)k} = a_{1(j+1)k} + a_{i1k} + a_{i(j+1)1} - a_{i11} - a_{1(j+1)1} - a_{11k} + a_{111}$

Therefore:

$$a_{ijk} - a_{i(j+1)k} = a_{1jk} - a_{1(j+1)k} + a_{ij1} - a_{i(j+1)1} + a_{1(j+1)1} - a_{1j1}$$

or

$$a_{1j1} + a_{1(j+1)k} + a_{i(j+1)1} + a_{ijk} = a_{1jk} + a_{1(j+1)1} + a_{ij1} + a_{i(j+1)k} \tag{3.73}$$

Tables 3.27 and 3.28 illustrate rectangle R1 for j being odd and even, respectively. An example of this construction for m = 5 is depicted in Table 3.29.

Table 3.27: Rectangle R1 for Case 3.29.2, j odd

	2					j	j+1				m+1
1	m+1		3		4	*1*	*k*				k-1
	3	k								k-1	2
	k	3	4						k-1	2	
		4	5						2		
				...			k-1				
					k-2	k-1	2				
i					k-1	*k*	*1*				
					2		k+1				
			k-1					...			m
		k-1	2						m-1	m	
	k-1	2							m	m+1	
m+1	2							m			m+1

Table 3.28: Rectangle R1 for Case 3.29.2, j even

	2						j	j+1			m+1
1	m+1		3				*1*	*k*			
	2	3		4		5					
	3	4	2								k-1
		2	4	5						k-1	
	4		5	6							
					...			k-1			
	5					k-2	k-1				
i						k-1	*k*	*1*			
								k			
				k-1					...		
			k-1							m	m+1
m+1		k-1								m+1	2

Table 3.29: Rectangle R1 for m=5 for Case 3.29.2, j odd & even resp.

	2		4		6
1	2		*1*	*5*	4
	3	6		4	2
	6	3	4	2	
4		4	*5*	*1*	
	4	2		6	
6	2				6

	2	3			6
1	6	*1*	*4*		
	2	3			
3	3	*4*	*1*		5
			5	2	
			2	5	6
6		5		6	2

Using $m+1$ elements, we illustrate that m (at least) top-right to bottom-left diagonals out of $2m+1$ can be completed. The rest can be completed using elements $m+2, ..., 2m+2$. The completion can be performed identically for both $(m+1, m+1, n)$ rectangles. The special case of $j = m+1$ is treated accordingly.

Case 3.29.3 *We consider the differences $a_{233} - a_{333}, ..., a_{(m+1)22} - a_{(m+1)22}$. In general, given that (3.68) is true for $a_{ijk}, (i, j, k) \in Q$, we prove that (3.68) is true for $a_{(i+1)jk}, (i+1, j, k) \in Q$ $(m+2 \equiv 2)$. According to (3.68):*

$a_{ijk} = a_{1jk} + a_{i1k} + a_{ij1} - a_{i11} - a_{1j1} - a_{11k} + a_{111}$

$a_{(i+1)jk} = a_{1jk} + a_{(i+1)1k} + a_{(i+1)j1} - a_{(i+1)11} - a_{1j1} - a_{11k} + a_{111}$

Therefore:

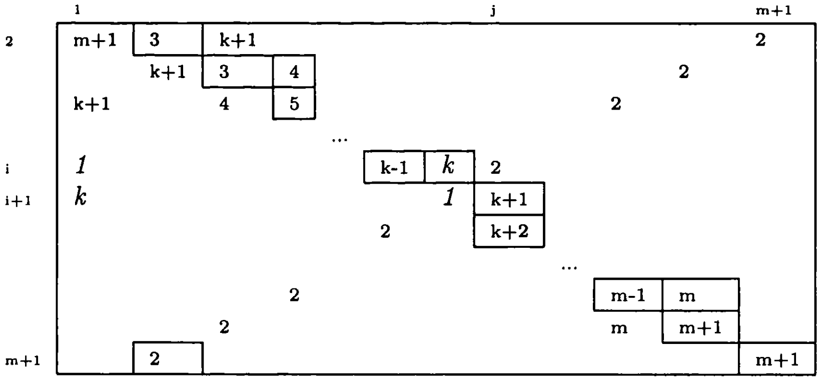
$$a_{ijk} - a_{(i+1)jk} = a_{i1k} - a_{(i+1)1k} + a_{ij1} - a_{(i+1)j1} + a_{(i+1)11} - a_{i11}$$

or

$$a_{i11} + a_{ijk} + a_{(i+1)1k} + a_{(i+1)j1} = a_{i1k} + a_{ij1} + a_{(i+1)11} + a_{(i+1)jk} \tag{3.74}$$

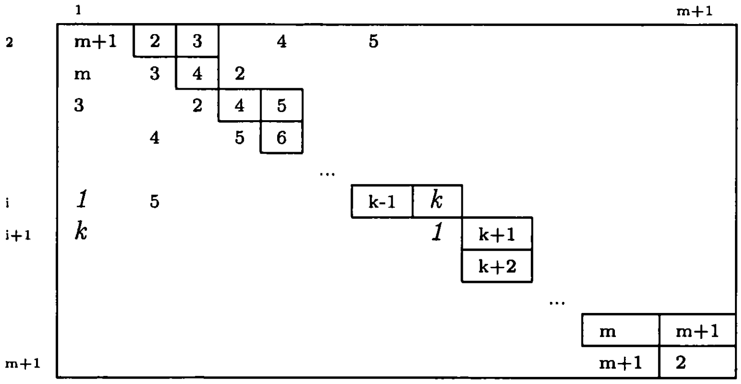
Tables 3.30 & 3.31 illustrate rectangle R1 for j being odd and even, respectively. An example of this construction for $m = 5$ is illustrated in Table 3.32.

Table 3.30: Rectangle R1 for Case 3.29.3, i odd



	1							j				m+1
2	m+1	3	k+1									2
		k+1	3	4							2	
	k+1		4	5						2		
					...							
i	*1*					k-1	*k*	2				
i+1	*k*						*1*	k+1				
						2		k+2				
									...			
				2						m-1	m	
			2							m	m+1	
m+1		2										m+1

Table 3.31: Rectangle R1 for Case 3.29.3, i even



	1											m+1
2	m+1	2	3		4		5					
	m	3	4	2								
	3		2	4	5							
		4		5	6							
						...						
i	*1*	5					k-1	*k*				
i+1	*k*							*1*	k+1			
									k+2			
										...		
											m	m+1
m+1											m+1	2

Using $m+1$ elements, we illustrate that m (at least) top-right to bottom-left diagonals out of $2m+1$ can be completed. The rest can be completed using elements $m+2, ..., 2m+2$. The completion

Table 3.32: Rectangle R1 for m=5 for Case 3.29.3, i odd & even resp.

	1			4		6
2	6	3	5			2
3	*1*	5	3	*4*	2	
	4			*1*	5	
			2		6	
6		2				6

	1				5	6
	6	2	3		4	
		3	4	2		
4	*1*		2	4	*5*	
	5	4			*1*	6
6	4				6	2

can be performed identically for both $(m+1, m+1, n)$ rectangles. The special case of $i = m+1$ is treated accordingly.

It remains to show that (3.68) is true for all $a_{ijk}, (i,j,k) \notin Q$. It is obvious that (3.68) holds if at least one index is equal to 1. For all other cases we have to prove that:

$$a_{ijk} = a_{1jk} + a_{i1k} + a_{ij1} - a_{i11} - a_{1j1} - a_{11k} + a_{111}$$

or

$$a_{11k} + a_{1j1} + a_{i11} + a_{ijk} = a_{111} + a_{1jk} + a_{i1k} + a_{ij1} \tag{3.75}$$

We examine a number of collectively exhaustive cases, each time illustrating rectangle R1 alone, which will give the coefficients of the r.h.s. of (3.75). Table 3.33 provides an overview of these cases.

Table 3.33: Remaining cases to be examined

	1	2 … m+1	m+2 … n
1			
2 m+1		2,3 3 3,4 4 Cases 3.29.7 & 3.29.8 … m,m+1 m+1 2 m+1,2	Case 3.29.6
m+2 m		Case 3.29.5	Case 3.29.4

Case 3.29.4 $i, j \in \{m+2, ..., n\}$, $k \in \{2, .., n\}$.

Assume w.l.o.g. that $i, j = m+2$. Consider rectangle R1, as illustrated in Table 3.34. No violation of the Latin square structure occurs, if $k > m+1$ or $k = 2$. R1 has $m+3$ out of $2m+3$ diagonals filled using $m+2$ elements (i.e. $1, ..., m+1$ & k). If $k \in \{4, ..., m\}$, it is enough not to fill the diagonal previously fixed with element k, to obtain a non-violated Latin square structure. In this case, R1 has $m+2$ out of $2m+3$ filled using $m+1$ elements. Finally, if $k = 3$(or $k = m+1$), the two diagonals filled using $3(m+1)$ can be filled using element $m+2$, which implies using $m+2$ elements to cover $m+3$ diagonals.

In all cases, based on Theorem 3.28 and the fact that $2m+2 \leq n$, R1 and the associated R2 are identically completable to a Latin square of order n. Note also that the corresponding integer points belong to $P_{LS}^{X(Q)}$. An example of the construction for $m = 5$, $i, j = 7$ & $k = 4$ is depicted in Table 3.35 .

Table 3.34: Rectangle R1 for Case 3.29.4

	1								m+1	m+2
1	*1*	m+1		3		4		5		*k*
	m+1	2	3		4		5		6	
		3	4	2				6		m-1
	3		2	4	5		6		m-1	
		4		5	6			m-1		m
						...			m	
		5		6			m-1	m		m+1
	5		6		m-1		m	m+1	2	
m+1		6		m-1				2	m+1	3
m+2	*k*		m-1		m		m+1		3	1

Table 3.35: Rectangle R1 for m=5, Case 3.29.4

	1			4		6	7
1	*1*	6		3			*4*
	6	2	3				5
		3	4	2		5	
	3		2	4	5		6
				5	6	2	
6			5		2	6	3
7	*4*	5		6		3	*1*

Case 3.29.5 $i \in \{m+2,...,n\}$, $j \in \{2,...,m+1\}, k \in \{2,..,n\}$.

Assume w.l.o.g. that $i = m+2$ *and* $j = 4$. *If* $k \in \{m+2,...,n\}$, *the completion of* $m+2$ *out of* $2m+2$ *diagonals of rectangle R1 using* $m+2$ *elements is implemented as shown in Table 3.36. If* $k \in \{4,...,m\}$, *but the Latin square structure is retained without any violation, it is sufficient to leave empty the diagonal filled by* k. *This will leave* $m+1$ *diagonals completed, the rest* $m+1$ *being filled with elements* $m+2,...,2m+2$. *If* $k = 3$ ($k = m+1$) *element* k *must be deleted from two diagonals, which will be both filled using element* $m+2$, *thus leaving* m *diagonals to be completed by elements* $m+3,...,2m+2$.

In the case that inserting element k *at cell* $(1, j)$ *violates the Latin square structure (e.g.* $k = 4$ *in Table 3.36), the choice of* p *variables belonging to* Q *must be slightly altered. For our example* ($k = 4$), *the construction is depicted in Table 3.37. Further adjustments of the chosen* p *variables are necessary to handle the cases where* $k = 2, 3, m+1$.

Table 3.36: Rectangle R1 for Case 3.29.5, k=m+2,...,n

	1			4					m+1
1	*1*	m+1		*k*					
	m+1	2	3		4		5		6
		3	4	2				6	
	3		2	4	5		6		m-1
		4		5	6			m-1	
						...			
		5		6			m-1	m	
	5		6		m-1		m	m+1	2
m+1		6		m-1				2	m+1
m+2	*k*		m-1	*1*	m		m+1		3

Case 3.29.6 $i \in \{2,...,m+1\}$, $j \in \{m+2,...,n\}$, $k \in \{2,..,n\}$.

Table 3.37: Rectangle R1 for Case 3.29.5, k=4

	1			4					m+1
1	*1*	m+1		*4*				5	
	m+1	2	3						6
		3	4			5		6	
	3			5	2				
				2	5	6			
			5		6	7			
							...		
	5		6					m	m+1
m+1		6							2
m+2	*4*			*1*					3

This case is symmetric to Case 3.29.5, in terms of treating the special subcases. The main points are again that element 3 $(m+1)$ *is used to complete two diagonals, which, if necessary, can be completed using element* $m+2$ *instead. The choice of* p *cells again has to be modified whenever the Latin square structure is violated by inserting element* k *at cells* $(1, j)$ *and* $(i, 1)$.

Case 3.29.7 $i, j \in \{2, ..., m+1\}$, $k \in \{2, .., n\}$: $(i, j, k) \notin Q$ *for all* k, *i.e. cell* (i, j) *is not used by* Q.

Assume w.l.o.g. that $i = 2$, $j = 4$. *If* $k \in \{m+2, .., n\}$, *R1 is constructed in the usual way, as illustrated in Table 3.38.* $m+2$ *out of* $2m+1$ *diagonals are filled using* $m+2$ *elements. If* $k \in \{4, ..., m\}$, *but the Latin square structure is retained without any violation, it is sufficient to leave empty the diagonal filled by* k, *since* m *empty diagonals can be filled with elements* $m+2, ..., 2m+2$. *If* $k = 3$ $(k = m+1)$, *two diagonals are left empty; however, element* $m+2$ *can be used to fill both.*

If a violation occurs, rectangle R1 is altered by considering different p *variables belonging to* Q. *As an example, the case of* $k = 3$ *is treated as shown in Table 3.39.*

Table 3.38: Rectangle R1 for Case 3.29.7, k=m+2,...,n

	1			4					m+1
1	*1*	m+1		*k*				5	
2	*k*	2	3	*1*	4		5		6
		3	4	2				6	
	3		2	4	5		6		m-1
		4		5	6			m-1	
						...			
		5		6			m-1	m	
	5		6		m-1		m	m+1	2
m+1		6		m-1				2	m+1

Case 3.29.8 $i, j \in \{2, ..., m+1\}$, $k \in \{2, .., n\}$: $(i, j, k) \in Q$ *for some* k.

Assume w.l.o.g. that $i = 3$, $j = 3$, *i.e. a cell used by the odd cycle. If* $k \in \{m+2, .., n\}$, *rectangle R1 is depicted in Table 3.40. If* $k \in \{2, .., m+1\}$ *the diagonal(s) filled with* k *is omitted, the special cases of* $k = 3, m+1$ *being handled as before. No violation of the Latin square structure can occur, since* $(i, j, k) \notin Q$. *For example, it is easy to check that* k *cannot be* 3 *or* 4 *in Table 3.40, since rectangle R1 would then have* $x_{333} = 1$ *or* $x_{334} = 1$ *and* $(3, 3, 3)$, $(3, 3, 4) \in Q$.

It remains to show that $\pi > 0$. Consider rectangles R1 and R2 as shown in Tables 3.41 & 3.42. Both are identically completable to Latin squares of order n. Rectangle R1 represents a point x_1,

Table 3.39: Rectangle R1 for Case 3.29.7, k=3

	1			4					m+1
1	*1*	m+1		*3*		4		5	
2	*3*	2		*1*	4				6
			3	4		5		6	
			4	5	2				
		4		2	5	6			
	4		5		6	7			m+1
							...		
	5		6					m	m+1
m+1		6				m+1		m+1	2

Table 3.40: Rectangle R1 for Case 3.29.7, k=m+2,...,n

	1		3						m+1
1	*1*	m+1	*k*	3		4		5	
	m+1	2	3		4				6
3	*k*	3	*1*	4		5		6	
	3		4	5	2				7
		4		2	5	6		7	
	4		5		6	7			m
							...		
	5		6		7			m	m+1
m+1		6		7		m		m+1	2

which does not belong to face $P_I^{X(Q)}$, since only $p-1$ variables belonging to Q are set to 1. Therefore, point x^1 satisfies the (strict) inequality $a^Tx^1 < a_0$. Rectangle R2 represents a point x^2 belong to $P_I^{X(Q)}$ (since $x^2_{222} = 1$), therefore $a^Tx^2 = a_0$. It follows that $a^Tx^1 < a^Tx^2$, which by cancelling out terms gives:

$$a_{111} + a_{112} + a_{212} + a_{221} < a_{112} + a_{121} + a_{211} + a_{222}$$

Recall also that, by Definition (3.70):

$$\pi = a_{112} + a_{121} + a_{211} + a_{222} - (a_{111} + a_{112} + a_{212} + a_{221})$$

It follows that $\pi > 0$.

Table 3.41: Rectangle R1 not belonging to Q

	1	2							m+1
1	*1*	*2*		3				5	
2	*2*	*1*	3		4		5		6
		3	4	2				6	
	3		2	4	5		6		m-1
		4		5	6			m-1	
						...			
		5		6			m-1	m	
	5		6		m-1		m	m+1	2
m+1		6		m-1				2	m+1

The proof is complete. ■

Table 3.42: Rectangle R2 belonging to Q

	1	2							m+1
1	*2*	*1*		3				5	
2	*1*	*2*	3		4		5		6
		3	4	2				6	
	3		2	4	5		6		m-1
		4		5	6			m-1	
						...			
		5		6			m-1	m	
	5		6		m-1		m	m+1	2
m+1		6		m-1				2	m+1

3.8 Concluding remarks

This chapter has provided a polyhedral characterisation of the OLS problem, in terms of establishing the dimension of the associated polytope and of proving that certain inequalities are facet-defining. The rank of the constraint matrix A has been identified by exploiting embedded $3AAP_n$ substructures. Certain irregularities of the OLS polytope have been surpassed, as for example the difficulty of exhibiting a trivial point of P_I for all n. This have been achieved by partially illustrating points, which are essentially isotopic to a feasible point of P_I. The inherent symmetry of the problem has been exploited via the interchange operator and by focusing only on non-symmetric cases. Our approach has also revealed a new class of facets for the Latin square, or $3PAP_n$, polytope, arising from odd-holes of the associated intersection graph. Apart from this theoretical contribution, we have also developed results that have algorithmic implications, in the form of separation algorithms for clique, antiweb and odd-cycle inequalities. All our separation schemes require a linear number of steps, with respect to the number of variables. These algorithms can be directly incorporated within a *Branch & Cut* scheme and this is precisely the topic addressed in the next chapter.

Chapter 4

Integer Programming algorithms

This chapter presents algorithms for solving the OLS problem using Linear and Integer Programming. The most motivating instance of the OLS problem is the infeasible case of $n = 6$. Proving infeasibility for this instance via Integer Programming, apart from being one of the aims of our research, demonstrates the benefits of using IP to solve this particular problem. Infeasibility was formally proved for the first time by Tarry ([81]), essentially by exhaustive enumeration. Shorter proofs have also been proposed, for example in [80]. This chapter presents an alternative proof, based solely on Linear Programming. The proof reduces the solution space of the original problem by incorporating a method for eliminating symmetries.

Computer-based methods have been successfully applied to answer open questions in finite algebra, the most characteristic example being the proof of the non-existence of a finite projective plane, or equivalently of a set of 9 MOLS, of order 10 ([58]). This work also explores the potential of algorithmic methods to answer theoretical questions related to Latin Squares. The innovative aspect of our approach is that this is implemented via mathematical programming. In particular, the non-existence of an OLS structure of order 6 is shown by solving exactly 12 linear programs. This IP model is presented in Section 4.1. A symmetry breaking mechanism, resulting in a reduction of the solution space, is described in Section 4.2. The actual proof of infeasibility for $n = 6$ is given in Section 4.3, where it is shown that 12 integer programming problems are infeasible because of the infeasibility of the corresponding linear programming problems. The infeasibility of each LP is demonstrated by exhibiting dual values, which in conjunction with Farkas' lemma ([68, Theorem 2.7]), illustrate the result.

The OLS problem for general n can be viewed as a feasibility problem, i.e. as the problem of identifying a pair of OLS of order n. Our work adopts exactly this perspective instead of treating OLS as an optimization problem. This conforms also to the applications of OLS (Section 1.4), which normally ask for a single, rather than an optimal, solution. In order to solve this problem, a "*Branch & Cut*" algorithm has been devised and is presented in this chapter.

Integer programming is simply the extension of linear programming to problems, where a subset of the variable set is required to be integer. *Pure* integer programs define the class of integer programs, where all problem variables must be integer. A special case of integer programs are the *binary* or $0-1$ integer programs (BIP), which involve only $0-1$ variables. Clearly, the OLS problem falls within this last category.

The study of integer programs was motivated by the success of the simplex method in solving linear programs. Extending this method to $0-1$ integer programs essentially partitions the original problem into subproblems, formed recursively by setting variables to 0 or 1. This is apparently an enumeration scheme, usually named "*Divide & Conquer*" or "*Branch & Bound*". The first term emanates from the computer science literature, whereas the second has been introduced by the mathematical programming community. As noted in [68, Chapter II.4], the first Branch & Bound algorithm for general integer programs was presented in [59], although the approach was broadly adopted after the first such algorithm appeared for the Travelling Salesman Problem (TSP) in [61]. Another version of the enumerative approach, specifically for $0-1$ programs, is the *implicit enumeration* algorithm ([7]).

A Branch & Bound algorithm can be improved by strengthening the LP formulation of the problem. This amounts to adding extra inequalities, generated by enforcing the fact that certain variables must be integer. A systematic procedure for generating all valid inequalities for an integer program is presented in [45] (see also [68, Section II.1-2]). This celebrated result initiated the theory of valid inequalities and motivated the development of the "*Branch & Cut*" algorithmic approach for solving integer programs. This generic scheme adds valid inequalities to the LP of each subproblem in Branch & Bound. These inequalities are also called "cuts" because they are normally added only if violated by the current fractional solution of the LP-relaxation. Again, the TSP has been the problem where one of the first Branch & Cut codes was applied ([66]). Because of the number of valid inequalities being enormous, a criterion for selecting the most prominent ones is required. Polyhedral theory has provided a natural characterisation of the strongest possible valid inequalities or facets (see also Section 3.1). Hence, several Branch & Cut codes incorporating families of, general purpose or problem specific, valid inequalities have been developed in the last decades. Numerous successful applications of this approach to both industrial and theoretical problems can be found in the mathematical programming literature (e.g. [68]). Especially set-packing and set-partitioning problems have attracted considerable attention, exactly because of their generic structure. The literature for set-partitioning includes algorithms for crew scheduling ([50]), and for the axial 3-index assignment problem ([11]). These algorithms use valid inequalities induced by cliques and odd-holes of the problem's intersection graph.

Section 4.4 presents the components of the Branch & Cut algorithm for the OLS problem. These components include a problem-specific preprocessor, a specialised branching mechanism, which uses *Special Ordered Sets of type I*, and separation algorithms for families of valid inequalities. The implementation details are analysed in Section 4.5. The computer code uses advanced features provided by the XPRESS-MP software ([31]) for branching, LP-solving and cut management. Finally, computational results are discussed in Section 4.6. These results measure different branching rules, certain strategies for cut addition and the quality of various families of cutting planes. The test problems used are the instances of the OLS problem for orders up to 12. Nevertheless, the case of $n = 6$ remains the most significant benchmark, since the entire solution space must be searched in order for infeasibility to be proved.

4.1 Two Integer Programming models for OLS

We illustrate again the IP model for OLS used in the previous chapters. The only difference is that the objective function is simply the sum of all the variables. This objective function results in every feasible solution being also optimal and relates to the fact that the OLS problem is treated as a

Table 4.1: A pair of *OLS* of order 4

0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

feasibility problem. For the purposes of this study, we will call this model $(IP1)$.

$$
\begin{aligned}
IP1 \quad &: \\
&\max \sum \{x_{ijkl} : i \in I, j \in J, k \in K, L \in L\} \\
\text{subject to} \quad &: \\
&\sum \{x_{ijkl} : i \in I, j \in J\} = 1, \forall k \in K, l \in L && (4.1)\\
&\sum \{x_{ijkl} : i \in I, k \in K\} = 1, \forall j \in J, l \in L && (4.2)\\
&\sum \{x_{ijkl} : i \in I, l \in L\} = 1, \forall j \in J, k \in K && (4.3)\\
&\sum \{x_{ijkl} : j \in J, k \in K\} = 1, \forall i \in I, l \in L && (4.4)\\
&\sum \{x_{ijkl} : j \in J, l \in L\} = 1, \forall i \in I, k \in K && (4.5)\\
&\sum \{x_{ijkl} : k \in K, l \in L\} = 1, \forall i \in I, j \in J && (4.6)\\
&x_{ijkl} \in \{0,1\} \forall i \in I, j \in J, k \in K, l \in L
\end{aligned}
$$

The integer solutions to this model correspond to pairs of OLS of order n. In Chapter 2, we defined the convex hull of integer points of $(IP1)$ as the polytope P_I. Throughout this chapter, polytope P_I will be called P_{OLS}, i.e. $P_{OLS} = conv\{x \in \{0,1\}^{n^4} : Ax = e\}$, where A is the constraint matrix of (4.1)-(4.6) and $e = (1,...,1)^T \in R^{6n^2}$. It is not difficult to see that each integer feasible vector must have exactly n^2 components equal to 1.

Assume that, alternatively, we wish to formulate the problem of checking whether a certain Latin square L_1 has an orthogonal mate L_2. Square L_2 should be decomposable into n disjoint transversals, according to Theorem 1.2. Since L_1 is given, each transversal should include n specific cells of L_2, such that all the corresponding cells of L_1 contain the same value. Moreover, all n cells of each transversal of L_2 should contain each value $l \in L$ exactly once. Let $T_{k_0} = \{(i_0, j_0) \in I \times J :$ cell (i_0, j_0) of L_1 contains value $k_0\}$, for all $k_0 \in K$. For example, for the first Latin square of Table 4.1, $T_0 = \{(0,0),(1,1),(2,2),(3,3)\}$.

Let the binary variable y_{ijl} be 1 if value l appears in cell (i,j) of square L_2 and 0 otherwise. Since each value must appear exactly once in each row of L_2, it must hold that $\sum\{y_{ijl} : j \in J\} = 1$, for all (i,l). Two more analogous sets of constraints are revealed by interchanging the roles of the sets I, J, L. Finally, for each T_{k_0} and for each l, the constraint $\sum\{y_{i_0 j_0 l} : (i_0, j_0) \in T_{k_0}\} = 1$ must be satisfied, in order for square L_2 to be orthogonal to L_1. The IP model is the following:

$$\begin{aligned}
IP2 \quad &: \\
& \max \sum \{y_{ijl} : i \in I, j \in J, L \in L\} \\
\text{subject to} \quad &: \\
& \sum \{y_{ijl} : i \in I\} = 1, \forall j \in J, l \in L && (4.7) \\
& \sum \{y_{ijl} : j \in J\} = 1, \forall i \in I, l \in L && (4.8) \\
& \sum \{y_{ijl} : l \in L\} = 1, \forall i \in I, j \in J && (4.9) \\
& \sum \{y_{i_0 j_0 l} : (i_0, j_0) \in T_{k_0}\} = 1, \ \forall k_0 \in K, l \in L && (4.10) \\
& y_{ijl} \in \{0,1\} \forall i \in I, j \in J, l \in L
\end{aligned}$$

Constraints (4.7)-(4.9), together with the integrality constraints, define the Latin square polytope, analysed in [37]. Again, each integer feasible solution of this model is bound to have n^2 variables equal to 1 and, because of the form of the objective function, is also optimal.

4.2 Reducing the solution space

A naive proof of infeasibility for the OLS problem for $n = 6$ would be to prove the non-existence of an orthogonal mate for each of the approximately 8×10^9 Latin squares of order 6. However, in order for redundant search to be reduced, symmetrical subcases should be excluded. The following discussion exhibits, first, why we can reduce the number of squares to be checked to 12 and, secondly, how additional reduction of the solution space can be achieved. In other words, the following analysis reduces the solution space of the original problem by proving that any subproblems (squares) not examined are symmetric to a subproblem included in the reduced solution space.

The group-theoretical definition of *isotopy* is applicable to Latin squares, viewed as multiplication tables of quasigroups. Hence, two Latin squares are *isotopic* (or *equivalent*) if one can be obtained from the other by permuting its rows, columns and elements ([34, p.168]). Extending this concept to OLS, we call two distinct pairs of OLS *isotopic* if one can be derived from the other by applying certain permutations to the rows, columns, elements of the first and elements of the second square. In our notation, this is equivalent to permuting the sets I, J, K and L respectively. Obviously, there can exist up to $(n!)^3$ distinct Latin squares isotopic to a certain square L_1 and up to $(n!)^4$ distinct pairs of OLS isotopic to a certain pair L_1, L_2. The set of Latin squares of order n is separated into subsets of isotopic squares, called isotopy classes ([34, Chapter 4]).

A special category of Latin squares is the ones having their first row and column arranged in natural order, like the first square of Table 4.1. These Latin squares are called *reduced.* Observe that, by properly permuting the elements of sets K and L, we can have the cells of the first row of a pair of OLS containing the integers $0, ..., n-1$ in natural order, without violating the orthogonality condition. Given this arrangement of the first row, we can permute the elements of set $I \backslash \{0\}$ in such a way that the first column of square L_1 is also in natural order, i.e. having L_1 to be a reduced Latin square. A pair of OLS of this form is called *standardised* or *reduced* ([34, p.159]). Fixing these $3n-1$ cells already reduces the problem size by a factor of $(n!)^2 \cdot (n-1)!$ ([60]). Nevertheless, this fixing does not capture all possible isotopisms of the problem, i.e. there can be two reduced Latin squares, which are isotopic to each other. According to the previous argument, if one Latin square of an isotopy class has an orthogonal mate, all Latin squares of this class do.

Another, less intuitive, form of symmetry is that of *conjugacy,* also known as *parastrophy* ([34, Section 2.1]). Two Latin squares of the same order n are called *conjugate*, if one can be derived from the other by permuting the roles of the sets I, J, K (or L). For example, if we interchange the roles of sets I, J for a certain square, we derive its transpose. Since there can be 6 permutations of the 3 sets, each Latin square can possess up to 5 distinct conjugates ([34, Theorem 4.2.1]). The concept of conjugacy is extendable to pairs of OLS, where the number of conjugates for a certain pair can be up to 23, i.e. the 4! permutations of 4 sets reduced by 1. Again, if a certain Latin square has an orthogonal mate, all its conjugates do as well, since the role of the four sets in our formulation is purely conventional. In $(IP1)$, the constraints (4.1)-(4.6) remain intact if we swap any two of the sets I, J, K, L. The Latin squares belonging to an isotopy class, together with their conjugates, constitute a broader class, called *main.* It has been proved that the set of all Latin squares of order n can be partitioned into main classes, while each main class is a union of isotopy classes ([34, Theorem 4.2.4]). The set of Latin squares of order 6 is partitioned into 12 main classes. Hence, in order to prove the non-existence of a pair of OLS of order 6, it is sufficient to prove the non-existence of an orthogonal mate for a single reduced Latin square from each of the 12 classes. In other words, we need to solve 12 cases of $(IP2)$. Before proceeding to the proof, we discuss how further fixing of the two squares is possible.

Consider a random OLS structure of order n. The simplest isotopism is the interchange of the roles of any two elements of a single set, e.g. the swapping of rows 0 and 1. The *interchange operator* ($\longleftrightarrow$), introduced in Section 3.2, facilitates this process. Assume $u \in P_{OLS}$. Writing $u^1 = u(m_1 \longleftrightarrow m_2)_M$, where $M = I, J, K, L$, implies that point $u^1 \in P_{OLS}$ represents a new OLS structure derived from u by interchanging the roles of members m_1 & m_2 of set M. Similarly, to express that u^1 is the conjugate of u derived by interchanging the roles of sets I and J, we would write $u^1 = u(I \longleftrightarrow J)$.

Let $X_{ij}, Y_{ij}, i, j = 0, ..., n-1$, denote the cells of the squares L_1 and L_2, respectively, and $D_{X_{ij}}, D_{Y_{ij}}$ be the sets of allowed values for each cell. In the reduced form, we already have $X_{0j} = Y_{0j} = j$ and $X_{i0} = i$ for $i, j = 0, .., n-1$. Now consider cell Y_{10} and observe that its possible contents are $D_{Y_{10}} = \{2, ..., n-1\}$, i.e. $Y_{10} \neq 0, 1$, since $Y_{00} = 0$ and pair $(1, 1)$ already appears in position $(0, 1)$. Assume a standardised pair of OLS having $Y_{10} = w$, where $w \in \{3, .., n-1\}$, and let $u \in P_{OLS}$ be the corresponding $0-1$ vector. Construct the point

$$u^1 = u(2 \longleftrightarrow w)_L (2 \longleftrightarrow w)_K (2 \longleftrightarrow w)_J (2 \longleftrightarrow w)_I$$

and observe that $u_{1012} = 1$, i.e. $Y_{10} = 2$, and u represents a standardised pair of OLS. It follows that if a solution having $Y_{10} = w$, $w \in \{3, .., n-1\}$, exists, a solution having $Y_{10} = 2$ must also exist. Therefore, we can fix pair $(1, 2)$ in position $(1, 0)$, thus reducing the solution space by an additional factor of $(n-2)$. Using the same approach, it can be proved that $D_{Y_{20}} = \{1, 3\}$, i.e. if there exists a solution with $Y_{20} = w$, $4 \leq w \leq n-1$, there exists also a standardised solution with $Y_{20} = 3$, having also $Y_{10} = 2$.

In general, observe that $D_{Y_{i0}} = \{1, 3, 4, ..., i-1, i+1, ..., n-1\}$, for $i \in \{3, ..., n-1\}$. This implies that $0, 2, i \notin D_{Y_{i0}}$ because $Y_{00} = 0, Y_{10} = 2$ and pair (i, i) already appears in position $(0, i)$. Assume a pair of OLS having $Y_{i0} = w$, where $w \in \{i+2, .., n-1\}$, and let u be the corresponding $0-1$

vector. Construct the point

$$u^1 = u((i+1) \longleftrightarrow w)_L((i+1) \longleftrightarrow w)_K((i+1) \longleftrightarrow w)_J((i+1) \longleftrightarrow w)_I$$

and observe that $u_{i0i(i+1)} = 1$. Hence, if there exists a solution having $Y_{i0} = w, w \in \{i+2, .., n-1\}$, another solution with $Y_{i0} = i+1$ is bound to exist. Notice also that the OLS structure remains standardised. Given these reduced domains, observe that value $n-1$ appears only in $D_{Y_{(n-2)0}}$. Therefore, we can also set $Y_{(n-2)0} = n-1$. The final form of the OLS structure, after this preliminary variable fixing, is depicted in Table 4.2.

Table 4.2: Variable fixing and domain reduction

0	1	···	n-2	n-1
1				
2				
⋮				
i				
⋮				
n-2				
n-1				

0	1	···	n-2	n-1
2				
{1,3}				
⋮				
{1,3,4,..,i-1,i+1}				
⋮				
n-1				
{1,3,..,n-2}				

4.3 Proving infeasibility

In order to devise the following proof of infeasibility, we exclusively use the model $(IP2)$, applying also the variable fixing described in the previous section. This fixing is implemented by adding extra inequalities.

First, we illustrate the proof for the trivial case of $n = 2$. It is not difficult to observe that there are only two Latin squares of order 2, depicted in Table 4.3. These squares are clearly not orthogonal, a fact that implies the non-existence of a pair of OLS for $n = 2$. Notice that each of these squares can be transformed to the other by interchanging the two rows or columns, i.e. these squares are isotopic. To prove the non-existence of OLS for $n = 2$, it is sufficient to prove that only one of these isotopic squares does not possess an orthogonal mate.

We solve the model $(IP2)$ for $n = 2$, where equality constraints are replaced by '$\leq$' constraints. Let L_1 be the first square of Table 4.3. Adding the inequalities

$$y_{0jj} \geq 1, \forall j \in J \tag{4.11}$$

to $(IP2)$ ensures that $y_{000} = y_{011} = 1$. This variable fixing is imposed by the analysis of the previous

Table 4.3: The two Latin squares of order 2

0	1
1	0

1	0
0	1

section. The actual model is the following:

$$\begin{aligned}
&\max \sum \{y_{ijl} : i \in I, j \in J, L \in L\}\\
&\text{subject to:}\\
&\sum \{y_{ijl} : i \in I\} \leq 1, \forall j \in J, l \in L\\
&\sum \{y_{ijl} : j \in J\} \leq 1, \forall i \in I, l \in L\\
&\sum \{y_{ijl} : l \in L\} \leq 1, \forall i \in I, j \in J\\
&y_{00l} + y_{11l} \leq 1, \forall l \in L\\
&y_{01l} + y_{10l} \leq 1, \forall l \in L\\
&-y_{0jj} \leq -1, \ j \in J\\
&0 \leq y_{ijl} \leq 1, \forall i \in I, j \in J, l \in L
\end{aligned} \tag{4.12}$$

Note that the constraints $y_{00l} + y_{11l} \leq 1$ and $y_{01l} + y_{10l} \leq 1$, for all $l \in L$, correspond to variables sets T_0, T_1 of square L_1. Solving this LP model provides an optimal value of 2. Since a Latin square of order 2 requires exactly 4 variables set to 1, it follows that L_1 cannot be part of an OLS structure. Table 4.4 illustrates the vector of dual values for constraints (4.7)-(4.10) and (4.11), treated as inequalities in (4.12).

Table 4.4: Dual values for a reduced Latin square of order 2

Ineq. (4.7)		Ineq. (4.8)		Ineq. (4.9)		Ineq. (4.10)		Ineq. (4.11)	
(j,l)	Dual	(i,l)	Dual	(i,j)	Dual	(k_0,l)	Dual		Dual
(0,0)	1	(0,0)	0	(0,0)	1	(0,0)	1		
(0,1)	0	(0,1)	0	(0,1)	1	(0,1)	0		2
(1,0)	0	(1,0)	0	(1,0)	0	(1,0)	0		2
(1,1)	1	(1,1)	0	(1,1)	0	(1,1)	1		

According to the duality theorem of Linear Programming ([68]), if we multiply each inequality of (4.12) with its dual value and add all resulting inequalities, we should derive an inequality whose left-hand side is the sum of all the variables and whose right-hand side is the value of the objective function. Hence, the vector of dual values provides an algebraic illustration of infeasibility. For $n = 2$, the result is the following.

$$\begin{aligned}
(y_{000} + y_{100} \leq 1) + (y_{011} + y_{111} \leq 1) +\\
(y_{000} + y_{001} \leq 1) + (y_{010} + y_{011} \leq 1) +\\
(y_{000} + y_{110} \leq 1) + (y_{011} + y_{101} \leq 1) +\\
2 \cdot (-y_{000} \leq -1) + 2 \cdot (-y_{011} \leq -1) =\\
y_{000} + y_{001} + y_{010} + y_{011} + y_{100} + y_{101} + y_{110} + y_{111} \leq 2
\end{aligned}$$

For $n = 6$, and as exhibited in Table 4.2, variables $y_{000}, y_{011}, ..., y_{055}$ along with variables y_{102}, y_{405} must be set to 1. In addition, we must ensure that $D_{Y_{20}} = \{1,3\}$, $D_{Y_{30}} = \{1,4\}$ and $D_{Y_{50}} = \{1,3,4\}$. It is easy to check that the only variable needed to be set explicitly to 0 is y_{204}; the rest are forced to be 0 because of the orthogonality constraints (4.10) or because of the previous fixing of the cells

Table 4.5: A Latin square of order 6, belonging to the second main class

0	1	2	3	4	5
1	0	4	5	2	3
2	5	0	4	3	1
3	4	5	0	1	2
4	3	1	2	5	0
5	2	3	1	0	4

in the first column of L_2. Hence, the set of additional inequalities is the following:

$$\begin{aligned} & y_{0jj} \geq 1,\ j \in J \\ & y_{102} \geq 1 \\ & y_{405} \geq 1 \\ - \quad & y_{204} \geq 0 \end{aligned} \tag{4.13}$$

Let D denote the constraint matrix of the constraints (4.7)-(4.10). The convex hull of all feasible $0-1$ vectors, i.e. of all Latin squares of order n that are orthogonal to L_1, is the polytope $\hat{P}_{OLS} = conv\{y \in \{0,1\}^{n^3} : Dy = e\}$, where $e = (1,...,1)^T \in 4n^2$. Let also F be the constraint matrix of inequalities (4.13) and $h = (1,..,1,0)^T \in R^{n+3}$. The convex hull of all integer vectors of $\hat{P}_{OLS}$, which also satisfy (4.13), is the polytope $P_I = conv\{y \in \{0,1\}^{n^3} : Dy = e, Fy \geq h\}$. The analysis of Section 4.2 essentially proves that $\hat{P}_{OLS} = \emptyset$ if and only if $P_I = \emptyset$. The linear relaxation of this formulation defines the polytope $P_L = \{y \in [0,1]^{n^3} : Dy = e, Fy \geq h\}$. Another relaxation of P_I is obtained if the '$=$' constraints (4.7)-(4.10) are converted to '$\leq$' ones. This is the *set-packing* relaxation $\tilde{P}_I = conv\{y \in \{0,1\}^{n^3} : Dy \leq e, Fy \geq h\}$, its linear relaxation being $\tilde{P}_L = \{y \in [0,1]^{n^3} : Dy \leq e, Fy \geq h\}$. Define also $Z_I = \{\max y^T 1 : y \in P_I\}$. $Z_L, \tilde{Z}_I, \tilde{Z}_L$ are defined accordingly. It is easy to see that $P_I \subset \tilde{P}_I \subset \tilde{P}_L$ and $P_I \subset P_L \subset \tilde{P}_L$. It follows that:

$$Z_I \leq \tilde{Z}_I \leq \tilde{Z}_L \tag{4.14}$$

$$Z_I \leq Z_L \leq \tilde{Z}_L. \tag{4.15}$$

As noted in Section 4.1, $Z_I = n^2$, since every feasible solution is also optimal. Hence, if $\tilde{Z}_L = z < n^2$, it follows that $Z_I \leq \lfloor z \rfloor$ or $Z_I \leq n^2 - 1$. This implies that $P_I = \hat{P}_{OLS} = \emptyset$. In order to prove that there exists no pair of OLS for $n = 6$, and based on the arguments exhibited in Section 4.2, it is sufficient to show that $\tilde{Z}_L < n^2$, i.e. $P_I = \emptyset$, for a single representative of each of the 12 main classes. This is exactly the nature of the proof presented in this paper. The vector of dual values for each $\tilde{Z}_L < n^2$ provides multipliers for all inequalities of $\tilde{P}_L$, which illustrate algebraically that the objective function, i.e. the sum of all variables, is strictly less than n^2. Alternatively, by incorporating Farkas' lemma ([68]), this dual vector proves the infeasibility for P_L and therefore also for P_I.

As an example, consider the reduced Latin square of Table 4.5, which is the representative of the second main class of Latin squares of order 6. The LP solved for this Latin square is the following:

$$\begin{aligned}
&\max \sum\{y_{ijl} : i \in I, j \in J, L \in L\}\\
&\text{subject to:}\\
&\sum\{y_{ijl} : i \in I\} \leq 1, \forall j \in J, l \in L\\
&\sum\{y_{ijl} : j \in J\} \leq 1, \forall i \in I, l \in L\\
&\sum\{y_{ijl} : l \in L\} \leq 1, \forall i \in I, j \in J\\
&y_{00l} + y_{11l} + y_{22l} + y_{33l} + y_{45l} + y_{54l} \leq 1, \forall l \in L\\
&y_{01l} + y_{10l} + y_{25l} + y_{34l} + y_{42l} + y_{53l} \leq 1, \forall l \in L\\
&y_{02l} + y_{14l} + y_{20l} + y_{35l} + y_{43l} + y_{51l} \leq 1, \forall l \in L\\
&y_{03l} + y_{15l} + y_{24l} + y_{30l} + y_{41l} + y_{52l} \leq 1, \forall l \in L\\
&y_{04l} + y_{12l} + y_{23l} + y_{31l} + y_{40l} + y_{55l} \leq 1, \forall l \in L\\
&y_{05l} + y_{13l} + y_{21l} + y_{32l} + y_{44l} + y_{50l} \leq 1, \forall l \in L\\
&y_{0jj} \geq 1,\ j \in J\\
&y_{102} \geq 1\\
&y_{405} \geq 1\\
&y_{204} \leq 0\\
&0 \leq y_{ijl} \leq 1, \forall i \in I, j \in J, l \in L
\end{aligned}$$

This LP has been solved by XPRESS-MP ([31]). The value of the objective function $\tilde{Z}_L$ is 34.5 and the dual values are exhibited in Table 4.6. Since all variables y_{ijl} must be integer, it follows that $\sum\{y_{ijl} : i \in I, j \in J, L \in L\} \leq 34$. This implies that the Latin square of Table 4.5, along with all squares of the second main class, has no orthogonal mate.

The representatives from each main class of $n = 6$ are displayed in [34, p.130-137], where each square is associated with a unique, three digit, number, whose first digit denotes the main class it belongs to. For example, the square of Table 4.5 is the square 2.1.1. The squares used for this proof are the squares 1.1.1, 2.1.1, 3.1.1, 4.1.2, 5.1.2, 6.1.3, 7.1.3, 8.3.1, 9.2.1, 10.2.1, 11.2.1, 12.3.1. An important note is that the value of $\tilde{Z}_L$ is not the same for all 12 representatives and the dual values are not always as symmetric as the ones exhibited in Table 4.6. Moreover, certain of the squares in [34, pp.130-137] provided a fractional solution with a value of $\tilde{Z}_L = 36$, as for example square 8.1.1. Such squares are obviously not proper for showing infeasibility by only solving an LP (i.e. without branching).

For all squares used in this proof, a value $\tilde{Z}_L < 36$ is achievable only after fixing variables y_{102}, y_{405} and y_{204} to 1, i.e. simply fixing the first row of L_2 has not been sufficient. Hence, the reduction of the solution space, as presented in Section 4.2, has been crucial for the LP-based proof for the non-existence of OLS for $n = 6$.

4.4 A Branch & Cut algorithm for OLS

The Branch & Cut algorithm, or B&C for short, accepts as input the order n of OLS and either produces as output a singe pair of OLS of order n or proves that no such pair exists. Note that the OLS problem is feasible for every $n \in \mathbb{Z}_+ \backslash \{1, 2, 6\}$ ([60, Theorem 2.9]). As noted before, the

Table 4.6: Dual values for a reduced Latin square of main class 2

Ineq. (4.7)		Ineq. (4.8)		Ineq. (4.9)		Ineq. (4.10)		Ineq. (4.13)
(j,l)		(i,l)		(i,j)		(k_0,l)		
(0,0)	0	(0,0)	0	(0,0)	1	(0,0)	0	0
(0,1)	0	(0,1)	0	(0,1)	0	(0,1)	0	0
(0,2)	0	(0,2)	0	(0,2)	0	(0,2)	0	0
(0,3)	0	(0,3)	0	(0,3)	0	(0,3)	0	0
(0,4)	0	(0,4)	0	(0,4)	1	(0,4)	0	0
(0,5)	1	(0,5)	1	(0,5)	1	(0,5)	1	-3
(1,0)	0.5	(1,0)	0.5	(1,0)	0	(1,0)	0.5	
(1,1)	0.5	(1,1)	0.5	(1,1)	0	(1,1)	0.5	0
(1,2)	0.5	(1,2)	0.5	(1,2)	0	(1,2)	0.5	0
(1,3)	0.5	(1,3)	0.5	(1,3)	0	(1,3)	0.5	-3
(1,4)	0.5	(1,4)	0.5	(1,4)	0	(1,4)	0.5	
(1,5)	0	(1,5)	0	(1,5)	0	(1,5)	0	
(2,0)	0.5	(2,0)	0.5	(2,0)	0	(2,0)	0.5	
(2,1)	0.5	(2,1)	0.5	(2,1)	0	(2,1)	0.5	
(2,2)	0.5	(2,2)	0.5	(2,2)	0	(2,2)	0.5	
(2,3)	0.5	(2,3)	0.5	(2,3)	0	(2,3)	0.5	
(2,4)	0.5	(2,4)	0.5	(2,4)	0	(2,4)	0.5	
(2,5)	0	(2,5)	0	(2,5)	0	(2,5)	0	
(3,0)	0.5	(3,0)	0.5	(3,0)	0	(3,0)	0.5	
(3,1)	0.5	(3,1)	0.5	(3,1)	0	(3,1)	0.5	
(3,2)	0.5	(3,2)	0.5	(3,2)	0	(3,2)	0.5	
(3,3)	0.5	(3,3)	0.5	(3,3)	0	(3,3)	0.5	
(3,4)	0.5	(3,4)	0.5	(3,4)	0	(3,4)	0.5	
(3,5)	0	(3,5)	0	(3,5)	0	(3,5)	0	
(4,0)	0	(4,0)	0	(4,0)	1	(4,0)	0	
(4,1)	0	(4,1)	0	(4,1)	0	(4,1)	0	
(4,2)	0	(4,2)	0	(4,2)	0	(4,2)	0	
(4,3)	0	(4,3)	0	(4,3)	0	(4,3)	0	
(4,4)	0	(4,4)	0	(4,4)	1	(4,4)	0	
(4,5)	1	(4,5)	1	(4,5)	1	(4,5)	1	
(5,0)	0	(5,0)	0	(5,0)	1	(5,0)	0	
(5,1)	0	(5,1)	0	(5,1)	0	(5,1)	0	
(5,2)	0	(5,2)	0	(5,2)	0	(5,2)	0	
(5,3)	0	(5,3)	0	(5,3)	0	(5,3)	0	
(5,4)	0	(5,4)	0	(5,4)	1	(5,4)	0	
(5,5)	1	(5,5)	1	(5,5)	1	(5,5)	1	

IP model used is $(IP1)$. The algorithm is enumerative in the sense that the original problem is recursively divided into subproblems. If Q denotes the original problem, a collection of subproblems $\{Q^1,...,Q^w\}$ is a *division* of Q if $\bigcup_{s=1}^{w} Q^s = Q$. Such a division is called a *partition* if $Q_s \cap Q_t = \emptyset$ for $s,t = 1,...,w$, $s \neq t$. The division of Q is implemented essentially by setting one or more of the binary variables to 0 or 1.

The actual B&C algorithm always forms partitions. A search tree is formed, where the top node corresponds to the original problem and each other node uniquely corresponds to a subproblem. Each problem is always partitioned into two subproblems, therefore the search tree is binary. Let us introduce some terminology at this point (borrowed from [23, Section 3.1.2]). A *binary tree* consists of a set of nodes, each having at most two other nodes as *immediate successors*, also called *children*. The top node of the tree is its *root*. Every node, except for the root, is the immediate successor of exactly one node, called its *immediate predecessor* or *father*. If node Q^i belongs to the branch of the tree having node Q^j as its root, Q^i is a *successor* (or descendant) of Q^j and Q^j is a predecessor (or antecedent) of Q^i. The division into subproblems is also called *branching* because it creates branches in the search tree.

Before the search starts, the preliminary variable fixing indicated by Table 4.2 is performed. Let $I = J = K = L = \{0,...,n-1\}$. Variables x_{0jjj}, for $j \in J$, along with two extra variables $x_{1012}, x_{(n-2)0(n-2)(n-1)}$, are set to 1. Variables x_{i0il}, for $i \in I$, $l = (i+1),...,(n-1)$ are set to 0. In order to enforce the requirement that the cells of the first column of square L_1 must be in natural order, only variables x_{i0il}, for $l \in L$ must be allowed to take value 1. This is accomplished by setting variables x_{i0kl} to 0, for $i \in I, k \in K\backslash\{i\}, L \in L$. Equivalently, the constraints $\sum\{x_{i0il} : l \in L\} = 1$, for $i \in I$, could have been appended to the original IP model. Hence, the variables fixed before starting the search are:

$$\begin{aligned} x_{0jjj} &= 1, \text{ for } j \in J \\ x_{1012} &= 1 \\ x_{(n-2)0(n-2)(n-1)} &= 1 \\ x_{i0il} &= 0, \text{ for } i \in I, l = (i+1), ..., (n-1) \\ x_{i0kl} &= 0, \text{ for } i \in I, k \in K\backslash\{i\}, l \in L \end{aligned} \qquad (4.16)$$

Let $LP(Q^s)$ denote the LP-relaxation of subproblem Q^s. If the algorithm can establish that no further division of Q^s is required, the enumeration tree is said to be *pruned* at the node corresponding to Q^s. It is also common to state that the node, or Q^s, is pruned. Clearly, Q^s is pruned if $LP(Q^s)$ is infeasible or if an integer feasible solution is found. If all subproblems are pruned without an integer solution having been found, Q is declared infeasible.

At each node of the search tree, subproblem Q^s is first preprocessed in order to fix additional variables to 0 or 1, if possible. This process reduces the size of the subproblem, thus accelerating the solution of $LP(Q^s)$. Preprocessing may also establish that Q^s is infeasible without even having to solve $LP(Q^s)$. If preprocessing does not certify infeasibility, $LP(Q^s)$ is solved to optimality. If the optimal solution is integer, the node is pruned and the B&C algorithm terminates. If $LP(Q^s)$ is infeasible, the node is also pruned. Otherwise, valid inequalities that are violated by the fractional solution are identified and, if any, are added to $LP(Q^s)$. The process of solving $LP(Q^s)$ and adding violated cuts is repeated until $LP(Q^s)$ is infeasible or an integer solution is found or no more violated cuts can be found. It can be verified that this iterative process is bound to terminate (see also [45]), although after an exponential number of iterations. In practice, an upper bound, polynomial in n, is placed upon the number of iterations. Given that Q^s cannot be pruned, it is divided into two subproblems. Algorithm 4.1 summarises the entire procedure.

Algorithm 4.1

```
Preprocess $Q^s$;
if ($Q^s$ remains feasible)
    repeat
    {
        Solve $LP(Q^s)$ ;
        If the optimal solution is non-integer
        {
            Identify valid inequalities violated by the fractional solution ;
            Add the violated inequalities to $LP(Q^s)$ ;
        }
    }
    until ($LP(Q^s)$ is infeasible) or (an integer solution is found) or (no inequalities are added)
if ($LP(Q^s)$ is still feasible) and (no integer solution has been found)
    Partition $Q^s$ into two subproblems;
else    prune $Q^s$;
return;
```

The components of the B&C algorithm to be further analysed are the preprocessor, the branching mechanism, the approach for identifying violated cuts and the methods for solving $LP(Q^s)$.

4.4.1 Preprocessing

The aim of preprocessing each subproblem is to reduce the problem size by excluding variables because other variables have already been set to 0 or 1. During the search, variables are being explicitly set to 0 or 1 because of branching. This is implemented either by adding an extra equality constraint or, most commonly, by altering the variable bounds. Initially, all variables have a lower bound of 0 and an upper bound of 1. Setting the upper bound of a variable to 0 is equivalent to setting this variable to 0. Similarly, setting the lower bound of a variable to 1, fixes this variable to value 1. Hence, at each node/subproblem, certain variables have been explicitly fixed. Preprocessing uses this information to fix even more variables (see also [6, 78]).

Recall from Section 2.2 that the row set of the constraint matrix A is $R = (K \times L) \cup (I \times L) \cup (J \times L) \cup (I \times J) \cup (J \times K) \cup (I \times K)$, while its column set is $C = I \times J \times K \times L$. Each column appears, i.e. contains a non-zero entry, in exactly 6 rows. For example, column (i_0, j_0, k_0, l_0) appears in the rows (i_0, j_0), (i_0, k_0), (i_0, l_0), (j_0, k_0), (j_0, l_0), (k_0, l_0). It is not difficult to see that column $c \in C$ appears in row $r \in R$ if and only if $r \subset c$. Notice that we treat members of sets R and C as subsets of indices, therefore the expression $r \subset c$ is valid.

Whenever a variable x_c is set to 1, all variables having 2 or 3 indices in common with c are automatically forced to 0 in any feasible solution, i.e. a total of $6(n-1)^2 + 4(n-1)$ variables. The matrix columns associated with these variables, along with the column c, can be removed from the matrix without losing any information. For the same reason, all rows where column c appears, i.e. $\{r \in R : r \subset c\}$, can be also removed. Similarly, whenever a variable x_c is set to 0, column c can be removed from A. Another possibility is that there has remained a single variable x_c not set to 0 in a row r. Clearly, x_c will be 1 in any feasible solution, therefore x_c can be explicitly set to 1 and the above mentioned rows and columns can be deleted. If all rows have a single variable not set to 0, it is obvious that an integer feasible solution has been found. If a certain row has all its variables set to 0, the subproblem is infeasible. Note that setting a variable to 0 or 1 might have further repercussions, therefore this process is repeated until matrix A is no longer reducible. We emphasise that this variable fixing would have been implicitly implemented whenever the LP is solved. The main reason for explicitly deleting rows and columns of A is to reduce its size, therefore accelerating the solution of $LP(Q^s)$. It is also possible that infeasibility, or an integer solution, is detected without having to solve $LP(Q^s)$.

Any deletions of rows and columns at a certain node are inherited in all descendant nodes. Therefore, we denote as A^s the matrix at the node associated with subproblem Q^s. R^s and C^s are defined accordingly. For each Q^s, the lists F_0^s, F_1^s of variables already fixed to 0 and 1 at its predecessors, i.e. deleted, are also retained. Obviously, $F_0^s \cup F_1^s = C \backslash C^s$. Algorithm 4.2 describes the preprocessing steps in pseudo-code. All rows and columns are repetitively checked until no more changes are possible. Each iteration includes $O(n^4)$ steps, since each remaining row and column must be examined. Let t denote the iteration counter.

Algorithm 4.2 (Preprocessing)

```
$t \leftarrow 0$;    $C^0 \leftarrow C^s$;    $R^0 \leftarrow R^s$;
repeat
{
    $t \leftarrow t+1$;    $C^t \leftarrow C^{t-1}$;    $R^t \leftarrow R^{t-1}$;
    for all $c_0 \in C^t$
    {
        if $x_{c_0}$ is set to 1
        {
            $F_0^s \leftarrow F_0^s \cup \{d \in C^t : |c_0 \cap d| = 2 \text{ or } 3\}$;
            $F_1^s \leftarrow F_1^s \cup \{c_0\}$;
            $C^t \leftarrow C^t \backslash \{d \in C^t : |c_0 \cap d| \geq 2\}$;
            $R^t \leftarrow R^t \backslash \{r \in R^t : r \subset c_0\}$;
        }
        else if $x_{c_0}$ is set to 0
        {
            $F_0^s \leftarrow F_0^s \cup \{c_0\}$;
            $C^t \leftarrow C^t \backslash \{c_0\}$;
        }
    }
    for all $r \in R^t$
        if $\{c \in C^t : r \subset c\} = \{c_0\}$
        {
            $F_0^s \leftarrow F_0^s \cup \{d \in C^t : |c_0 \cap d| = 2 \text{ or } 3\}$;
            $F_1^s \leftarrow F_1^s \cup \{c_0\}$;
            $C^t \leftarrow C^t \backslash \{d \in C^t : |c_0 \cap d| \geq 2\}$;
            $R^t \leftarrow R^t \backslash \{r \in R^t : r \subset c_0\}$;
        }
}
until $(C^t = C^{t-1})$ and $(R^t = R^{t-1})$
$C^s \leftarrow C^t$;    $R^s \leftarrow R^t$;
return;
```

When Algorithm 4.2 is applied to Q^0, i.e. the original problem, it will delete all columns indicated by (4.16). It will also delete extra rows and columns for each of the variables initially set to 1. As an example, consider variable x_{1012}. The rows to be deleted are $(1,2) \in K \times L$ (eq. (4.1)), $(0,2) \in J \times L$ (eq. (4.2)),... and $(1,0) \in I \times J$ (eq. (4.6)). The deleted columns are $\{x_{i012} : i \in I\}$, $\{x_{1j12} : j \in J\}$, $\{x_{10k2} : k \in K\}$, $\{x_{101l} : l \in L\}$ and $\{x_{ij12} : i \in I, j \in J\}$, ..., $\{x_{10kl} : k \in K, l \in L\}$.

4.4.2 The branching mechanism

This section discusses the division of a subproblem Q^s into two subproblems. Recall that Q^s is partitioned only if there is an optimal fractional solution of $LP(Q^s)$. The simplest method for partitioning Q^s is to select a fractional variable $x_c \in (0,1)$ and form two subproblems, namely Q_0^s and Q_1^s, where x_c is set to 0 and 1 in Q_0^s and Q_1^s, respectively. This scheme is known in the IP literature as *variable dichotomy* ([68, Chapter II.4, Section 2]). Unless $x_c = 0$ in most feasible solutions, Q_1^s will contain significantly fewer feasible solutions than Q_0^s. In other words, setting x_c to 0 achieves no real progress and the set of feasible solutions in Q_0^s is almost identical to that of Q^s. Because of the extensive symmetries existing in the OLS problem, no variable can be signalled as unlikely to be 1 in a feasible solution. Therefore, variable dichotomy appears to be a mediocre choice for branching.

It seems better to perform branching in a way that approximately halves the solution space of Q^s. A method achieving this is commonly referred to as *Generalised Upper Bound* (GUB) *dichotomy.* Each constraint of the form $\sum_{c\in C_0} x_c = 1$, for $C_0 \subset C$, is called a GUB constraint. Notice that $(IP1)$ contains exclusively GUB constraints. The variable set of such a constraint is also said to form a *Special Ordered Set of type I*, or SOS-I for short ([13]). Formulation $(IP1)$ contains $6n^2$ SOS-I, each of cardinality n^2. Let S represent the set of variables appearing on the left-hand side of a particular equality. Exactly one variable in S will be 1 at any feasible $0-1$ vector. It follows that, for $S_1 \subset S$, the single variable of S set to 1 will be either in S_1 or in $S\backslash S_1$. It is easy to see that the problem can be partitioned into two subproblems, each defined by setting the variables of either S_1 or $S\backslash S_1$ to 0. Formally, Q^s is partitioned into Q^{s_1} and $Q^{s\backslash s_1}$, where variables $\{x_c : c \in S_1\}$ are set to 0 in Q^{s_1} and variables $\{x_c : c \in S\backslash S_1\}$ are set to 0 in $Q^{s\backslash s_1}$. Notice that variable dichotomy is a special case of GUB dichotomy, where $S_1 = \{x_c\}$. In order to ensure that the sets of feasible solutions of Q^{s_1} and $Q^{S\backslash S_1}$ are almost equally large, sets S_1 and $S\backslash S_1$ should be of approximately the same cardinality.

By recursively partitioning sets S_1 and $S\backslash S_1$, eventually a subproblem will be left with a single variable of S not set to 0, which is bound to be 1. If $|S_1| = |S\backslash S_1| = \frac{n^2}{2}$, a variable is set to 1 at most after $\lceil log_2 n^2 \rceil = \lceil 2log_2 n \rceil$ or $O(log_2 n)$ partitions (levels of the tree), since each equality constraint involves n^2 variables. Whenever a variable is set to 1, the partition proceeds by selecting another SOS-I, meaning another equality, whose left-hand side has at least one variable still not set to 0. It is not difficult to prove that, GUB dichotomy, compared to variable dichotomy, drastically reduces the depth of the search tree (see also [63, 68]). What remains to be clarified is how to select the SOS-I among the $6n^2$ ones available.

To facilitate the following discussion, let us examine the OLS problem independently of the actual formulation. As in Section 4.2, let $X_{ij}, Y_{ij}, i, j = 0, ..., n-1$, denote the cells of the squares L_1 and L_2 (also referred to as squares X, Y). Let $D_{X_{ij}}, D_{Y_{ij}}$ be the sets of allowed values (domains) for each cell (i, j). Suppose also that certain cells are initially fixed as indicated by Table 4.2. The search should select a certain cell and then select a pair to be assigned to this cell. It seems reasonable to initiate the search by fixing the remaining $(n-3)$ cells in the first column of Y, i.e. cells $Y_{20}, ..., Y_{(n-3)0}, Y_{(n-1)0}$. Once this have been done, the cells of the remaining $(n-1) \times (n-1)$ subsquare must be fixed.

There are two systematic methods for selecting the next cell to be examined. One is to examine all cells in a certain row (or column). Since the cells of the first row and column are already fixed, cell $(1, 1)$ is selected first, followed by cell $(1, 2)$ and so on, until cell $(1, n-1)$ is also assigned a pair. Once the first row of OLS is complete, the algorithm switches to the next row, i.e. cell $(2, 1)$ is selected and a pair $(2, l)$, $l \in D_{Y_{21}}$, is assigned to it. The last cell to be assigned a pair is $(n-1, n-1)$. Equivalently, we could proceed by fixing all elements in the first column, then in the second one and so on.

The alternative method is, having assigned a pair to cell (i, j), to switch to a pair (i', j') such that $i' \neq i$ and $j' \neq j$. This implies that we always select a cell in a different row and column. Roughly speaking, this method constructs the squares in a "diagonal" fashion. Notice that, since the first method fixes rows of the squares, it has to select each time a different value for a cell in the current row of both X and Y. This second method does not need to select a different value for X each time, since, pairwise, all cells are in different rows and columns. Therefore, it can set all cells of square X to the same value and set, according to the definition of OLS, all cells of square Y to pairwise different values. Hence, this method fixes transversals of square Y.

Each time a cell is assigned a pair, certain values are no longer valid in the domains of the remaining cells, since each pair must occur exactly once. These values can be then deleted from the domains of the remaining cells. A sensible criterion for branching is to select the next cell in a way that maximises the number of values removed from the domains of the remaining cells. Such an approach ensures the maximum reduction of the remaining solution space, therefore the minimum tree depth. Using simple counting arguments, we show that, under this criterion, it is better to fix transversals rather than rows. This branching heuristic actually arises from the literature of Constraint Programming ([83]), an approach to be further analysed in the next chapter.

Suppose that the search proceeds by fixing the remaining $(n-1)$ cells of the second row in both squares. Since $X_{10} = 1$, the values $0, 2, ..., n-1$, i.e. $n-1$ different values, will appear in the first row of square X. Setting $X_{1j_0} = k_0$, for $j_0 = 1, ..., n-1$, implies the deletion of value k_0 from the domains of all cells in the same column. These are cells $X_{i_0 j_0}$ for $2 \leq i_0 \leq n-1$. For $i_0 = k_0$, no domain member is removed, since having fixed $X_{i_0 0} = k_0$, $k_0 \neq 0$, implies that already $k_0 \notin D_{i_0 j_0}$. Hence, $(n-3)$ domain members are removed for each of the $(n-2)$ values k_0 that are different from 0. The case $k_0 = 0$ is an exception because 0 does not appear in the first column of any of the remaining rows. Hence, setting $X_{1j_0} = 0$ results in removing value 0 from all $(n-2)$ rows. In total, $(n-2) \cdot (n-3) + (n-2) = (n-2)^2$ domain values are removed from square X. Observe that the case of square Y is identical, given that the first column of Y has already been fixed. Thus, $(n-2) \cdot (n-3) + (n-2) = (n-2)^2$ domain values are also removed from square Y. The total number of values removed from the domains of the remaining cells is

$$\alpha(n) = 2 \cdot (n-2)^2$$

Table 4.7: Counting the number of values removed from square Y

0	1	$\cdots$	j_1	$\cdots$	j_0	$\cdots$
2						
$\vdots$						
i_1						
$\vdots$						
i_0					l_0	
$\vdots$						

Suppose that, alternatively, we fix a transversal of square Y, along with fixing the corresponding cells of square X to the same value 0. Value 0 will be removed from all the remaining cells of square X. It is easy to see that the number of those cells is $(n-1)^2 - (n-1) = (n-1)(n-2)$. For square Y, since pair $(0,0)$ already appears in cell $(0,0)$, values $1, .., n-1$ will appear in this first transversal. Setting $Y_{i_0 j_0} = l_0$ will remove value l_0 from all cells in row i_0 and column j_0. Having fixed the first row and column, however, implies that value l_0 has already been removed from exactly one cell of row i_1 and another cell of column j_1 (Table 4.7). Hence, for each $l_0 \neq 0$, value l_0 is removed from the domains of exactly $2 \cdot (n-3)$ cells. For $l_0 = 0$, $2 \cdot (n-2)$ domain members are removed, since value 0 appears only at cell $(0,0)$. The total number of domain values removed from square Y is $2 \cdot (n-2) + 2 \cdot (n-2) \cdot (n-3) = 2 \cdot (n-2)^2$. Hence the number of values removed from both squares is

$$\beta(n) = 2 \cdot (n-2)^2 + (n-1) \cdot (n-2)$$

Since $\beta(n) = a(n) + (n-1) \cdot (n-2)$, it follows that $\alpha(n) \leq \beta(n)$ for all $n \in \mathbb{Z}_+ \backslash \{0\}$. Therefore, fixing transversals appears to be computationally more efficient, a fact also supported by the experimental results of Section 4.6.

Let us now return to the main issue of how to select the SOS-I for partitioning a subproblem. First, we need to fix the first column of square Y. This implies selecting which variables of the form $x_{i_0 0kl}$ will be set to 1. It is not difficult to see that the SOS-I is the variable set of the constraints:

$$\sum\{x_{i_0 j_0 kl} : k \in K, l \in L\} = 1, \text{ for all } i_0 \in I \backslash \{0, 1, (n-2)\}, j_0 = 0 \tag{4.17}$$

which is a subset of constraints (4.6). However, $k = i_0$ since the first column of X has been fixed in natural order. This means that only variables $x_{i_0 0 i_0 l}$ must be considered; the rest have already been set to 0 by preprocessing. Hence, (4.17) becomes:

$$\sum\{x_{i_0 0 i_0 l} : l \in L\} = 1, \text{ for all } i_0 \in I \backslash \{0, 1, (n-2)\}$$

After having branched on all of these $(n-3)$ SOS-I, the first column of Y will have been fixed.

Next, if we wish to fix transversals, we need to fix all values k_0 in all rows i_0. Thus, for each $(i_0, k_0) \in I \times K$, we need to select which of the variables $x_{i_0 j k_0 l}$ will be set to 1. The SOS-I is the variable set of a constraint:

$$\sum\{x_{i_0 j k_0 l} : j \in J \backslash \{0\}, l \in L\} = 1, \text{ for all } i_0 \in I \backslash \{0\}, k_0 \in K$$

which is clearly the constraint set (4.5). Fixing, for example, value 0 in row 1 is implemented by recursively partitioning the variable set of equality $(1, 0) \in I \times K$ of (4.5). A feasible solution will be found after branching on, at most, $(n-1)^2$ SOS-I emanating from constraints (4.5). Since complete branching on a single constraint requires up to $O(log_2 n)$ levels, the maximum tree depth can be $O(n^2 log_2 n)$. If we wish, instead, to fix rows of the squares, the SOS-I should be selected from constraints (4.6). The maximum tree depth is again $O(n^2 log_2 n)$ and, according to the previous analysis, it can be no larger than the maximum tree depth achieved by fixing transversals.

For the branching strategy that fixes transversals of square Y, the SOS-I always has the form $\{x_{i_0 j k_0 l} : j \in J \backslash \{0\}, l \in L\}$. The special ordered sets are considered in increasing order first with respect to index k_0 and then with respect to index i_0. In other words, value k_0 is fixed in $n-1$ cells of square X before value $k_0 + 1$. Similarly, a certain value k_0 is fixed in row i_0 before being fixed in row $i_0 + 1$. Within a SOS-I, variables are ordered increasingly with respect first to index j_0 and then to index l_0. This implies fixing value k_0 in the first available cell of the current row with the smallest available value for square Y. For example, for fixing value 0 in row 1 of X, cell $(1,1)$ will be examined first. Since $Y_{01} = 1$ and $Y_{10} = 2$, value $l_0 = 3$ will be the first to be assigned to cell $(1,1)$ of square Y. Let us illustrate the above with an example.

Example 4.1 *For $n = 5$, the form of Table 4.8 is the same as Table 4.2. Assume that, after partitioning the initial subproblem, we are at a node Q^s of the search tree, where Y_{20} has been set to value 1 and, consequently, Y_{40} has been forced by preprocessing to take its single allowed value 3. The next step is to fix value 0 at row 1 of square X and also fix the corresponding cell of square Y to a value different from 0 (pair $(0,0)$ has appeared already). Table 4.9 illustrates the domains of all* *cells in row* 1 *of square Y.*

Table 4.8: A pair of OLS after preliminary variable fixing

0	1	2	3	4
1				
2				
3				
4				

0	1	2	3	4
2				
{1,3}				
4				
{1,3}				

Table 4.9: Allowed values at the first row

0	1	2	3	4
1				
2				
3				
4				

0	1	2	3	4
2	{3,4}	{1,3,4}	{1,4}	{1,3}
1				
4				
3				

The SOS-I at this subproblem is $\{x_{1j0l} : x_{1j0l} \in C^s\}$. *It is not difficult to see that the actual variables are* $\{x_{1103}, x_{1104}, x_{1201}, x_{1203}, x_{1204}, x_{1301}, x_{1304}, x_{1401}, x_{1403}\}$. *When this subproblem is partitioned, variables* $\{x_{1204}, x_{1301}, x_{1304}, x_{1401}, x_{1403}\}$ *will be set to* 0 *at its left child and variables* $\{x_{1103}, x_{1104}, x_{1201}, x_{1203}\}$ *will be set to* 0 *at its right child. This implies that the left child contains all feasible solutions where pairs* $(0,3), (0,4)$ *appear in cell* $(1,1)$ *or pairs* $(0,1), (0,3)$ *appear at cell* $(1,2)$. *Figure 4.1 depicts the branch of the search tree, whose top node is* Q^s. *At each node, the variables set to* 0 *(or forced to* 1*) are also shown.*

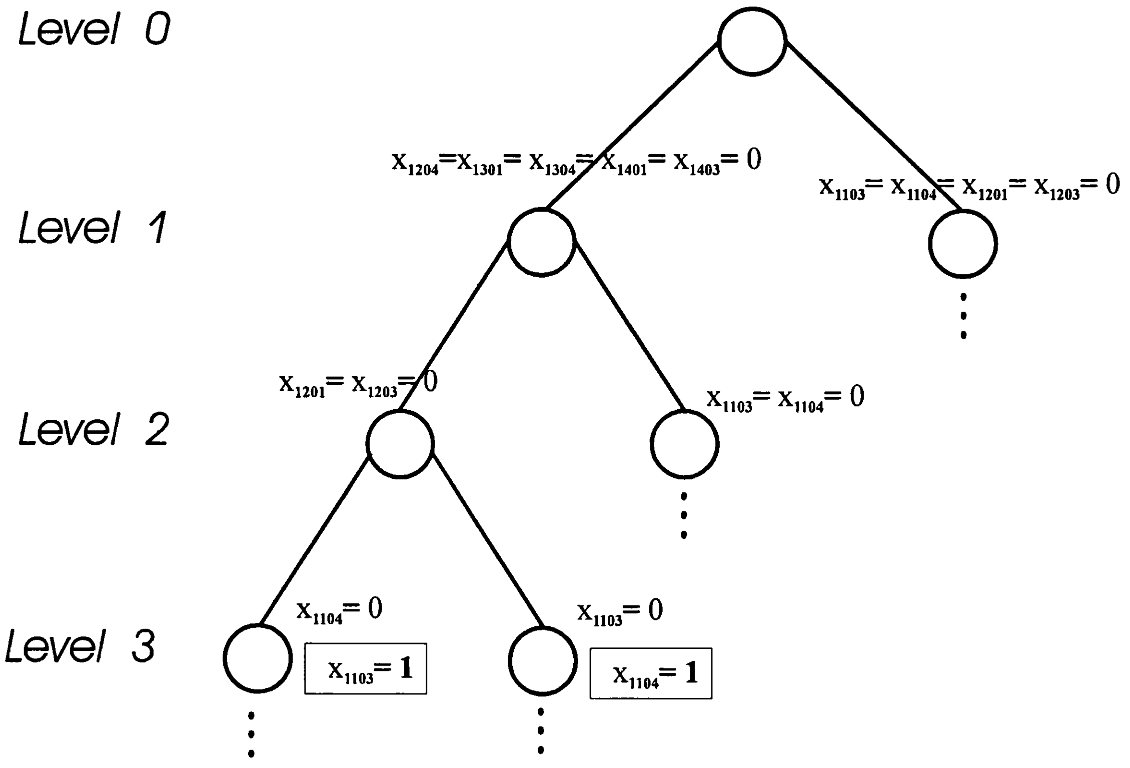


Figure 4.1: An example of the branching mechanism

The last issue to be clarified is which of the created subproblems is examined first. The B&C code applies *depth-first* search: if the current node Q^s is not pruned, one of its children is the next node to be examined. The left child is examined first. The right child will be examined only if the search in the branch defined by the left child is fruitless, i.e. when the B&C algorithm *backtracks* to Q^s. This option, apart from simplicity, has been selected mainly because the search is for a single feasible solution. Experience shows that feasible solutions tend to appear deep in the tree rather than at nodes near to the root ([68, p.358]).

4.4.3 Cutting planes

In Chapters 2 & 3, families of strong valid inequalities were identified for the OLS polytope P_{OLS}. Adding these inequalities to the LP-relaxation of the original problem Q or any subproblem Q^s results in a tighter problem formulation. The term "tighter" refers to the fact that $LP(Q^s)$ has a smaller solution space after adding extra inequalities. For general integer programming, tighter relaxations provide better bounds on the value of the objective function and therefore reduce the integrality gap, i.e. the gap between the LP-optimum and the optimal integer solution. This is not applicable in the OLS case, since we are only interested in a feasible solution or for a proof of infeasibility. Moreover, the objective function $\sum\{x_c : c \in C\}$ has optimal value n^2 for all $LP(Q^s)$, i.e. the duality gap is zero for all fractional LP solutions. The use of cutting planes aims solely at *(a)* identifying an infeasible subproblem Q^s at the highest possible tree level (especially for $n = 6$) or *(b)* adding sufficient faces in order for the LP to provide an integral vertex of the polytope.

As discussed elsewhere, the number of valid inequalities is prohibitively large to add them all at once. Cut addition must always ensure that the increase in the matrix size does not counteract the benefits achieved by reducing the solution space of $LP(Q^s)$. A first step towards this goal is the separation algorithms described in Chapter 3 for clique, lifted antiweb and lifted 5-hole inequalities. These algorithms provide a single valid inequality of a certain family, only when violated by the current LP-solution. They also run in linear time with respect to the number of variables, a fact that makes violated cuts comparatively cheap to generate. These families are problem-specific and define *global* cuts, in the sense that they are valid for any node throughout the search tree, since they are valid for the original problem Q ([50]). Another category of general purpose cutting planes is the *fractional Gomory cuts.* These are *local* cuts because they are valid only for the node, where they are generated. The B&C algorithm uses these cuts by applying the standard separation scheme, described next for the OLS formulation ($IP1$) ([68, p.367-370]).

Consider the IP:

$$max\{e^t x : Ax = e, x \in \{0,1\}^{n^4}\}$$

where A is a $6n^2 \times n^4$ matrix. Rewrite this IP as

$$max\{x_0 : x_0 = e^t x, Ax = e, x_0 \in \mathbb{Z}_+, x \in \{0,1\}^{n^4}\} \tag{4.18}$$

Supposing that an optimal basis for $LP(Q^s)$ has been obtained, (4.18) can be written as:

$$\begin{aligned} & max\ x_0 \\ & x_{B_i} + \sum_{j \in H} \bar{a}_{ij} x_j = \bar{a}_{i0}, \text{ for } i = 0, ..., 6n^2 \\ & x_{B_0} \in \mathbb{Z}_+, x_{B_i} \in \{0,1\} \text{ for } i = 1, ..., 6n^2, \ \ x_j \in \{0,1\} \text{ for } j \in H \end{aligned} \tag{4.19}$$

where $x_0 = x_{B_0}$, x_{B_i} for $i = 1, ..., 6n^2$ are the basic variables and x_j for $j \in H$ are the non-basic ones. Since the basis is primal and dual feasible, it holds that $\bar{a}_{i0} \geq 0$ for $i = 1, ..., 6n^2$ and $\bar{a}_{0j} \geq 0$ for $j \in H$. Suppose that there exists an i such that $\bar{a}_{i0} \notin (0, 1)$, i.e. variable x_{B_i} is fractional. The following proposition identifies the Gomory fractional cut associated with this variable.

Proposition 4.3 (Gomory fractional cut) *If* $\bar{a}_{i0} \in (0, 1)$ *then*

$$\sum_{j \in H} f_{ij} x_j \leq f_{i0} \tag{4.20}$$

where $f_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$, $f_{ij} = \bar{a}_{i0} - \lfloor \bar{a}_{i0} \rfloor$, *is a valid inequality for (4.18).*

Observe that inequality (4.20) is violated by the current LP-solution. It is easy to see that (4.20) can be written with respect to the basic variables, using the simplex tableau. Notice also that a Gomory fractional cut can be obtained for each fractional variable.

Further technical details of the cut addition process must be analysed. Notice first that the computational effort imposed by cut addition refers to generating the cut and, mainly, to re-optimising $LP(Q^s)$ in order to derive a new feasible solution, or prove that the amended $LP(Q^s)$ is infeasible.

First, the separation algorithms, in the form given in Chapter 3, add only one cut at a time. This approach might require numerous iterations of cut addition and re-optimisation before a viable outcome is achieved. A greedier approach of adding more the one violated inequality from each family sounds more appropriate. The natural question is what should be the maximum number of inequalities added from each family. Clearly, this upper bound should be a polynomial of n of rather small degree, e.g. a multiple of n or n^2. The number of cuts added at each iteration, in conjunction with the maximum number of iterations at each node, determines the "aggressiveness" of the cut strategy. Recall that the number of iterations should also be polynomial in n in order to ensure a polynomial time procedure for solving each subproblem. It follows that a B&C algorithm faces an obvious trade-off between adding more violated cuts at each node, hence reducing the number of subproblems, and posing more computational effort at each node. There is no unequivocal answer to this question. Normally, the answer is problem specific and requires computational experimentation. We discuss this topic further in Section 4.6.

A second aspect is the order in which families of valid inequalities are examined. As discussed in Sections 3.5 and 3.7.1, the separation algorithms for lifted antiweb and lifted 5-hole inequalities are correct only if no violated clique inequalities have been found. Correctness refers to the algorithms' ability to provide a violated cut, if such exists. This implies that inequalities induced by cliques of type II and III should be separated first and lifted antiweb and lifted 5-hole inequalities should be examined only if no violated clique inequalities arise. Notice, however, that separation of, say, lifted antiweb inequalities can still provide a violated cut, even though clique inequalities have not been examined. In practice, one should separate first the inequalities which are *(a)* more effective in pruning a node and *(b)* on average cheaper to generate. For example, if lifted 5-hole inequalities happen to be stronger in this sense, they should be the first to be added to the matrix, the rest of the families to be examined afterwards. Hence, regardless of the algorithmic correctness, one must still experiment with the order in which families of valid inequalities are separated. It is expected, however, that Gomory fractional cuts will be the weakest ones, exactly because they are local and encompass no knowledge about problem's structure.

Another critical aspect is the frequency of cutting planes addition. On the one hand, cutting planes can be added at every single node of the tree, as in [50]. This approach, apart from being computationally more expensive, does not guarantee more successful pruning, since the problem is not significantly different in each node. If, for example, a single variable is set to 0 for creating a subproblem (e.g. the nodes of Level 3 in Figure 4.1), this subproblem will be almost identical to its father. The other extreme is to add cutting planes only at the top node of the tree and simply retain them in all other nodes. The reasonable option lies between these two extremes ([6, 68]). Hence, cuts are added at the top node of the tree and then whenever a variable is forced to become 1 by preprocessing. A variable becoming 1 offers a practical criterion for detecting that the current subproblem is sufficiently different from its ancestors. Recall that, by branching on SOS-I, a variable is forced to value 1 at most every $\lceil 2 \cdot log_2 n \rceil$ levels of the tree. Another, more conservative, approach is to add cuts once after the first column is fixed and then every time a transversal is fixed, i.e. whenever $(n-1)$ variables are set to 1. Clearly, this takes place at most every $\lceil 2 \cdot (n-1) log_2 n \rceil$ tree levels.

Normally, all cutting planes added at a node are retained in all descendant nodes. This might eventually cause a drastic increase in the matrix size. Moreover, cuts added early in the tree are not necessarily useful in subsequent nodes. Therefore, a method for determining which cuts to retain must be devised. Dropping all cuts of a node is not practical either, because the LP-relaxation of its children must be tightened again from the beginning. An intermediate option, extensively used in recent B&C codes (see for example [5]), is to retain at descendant nodes only the cutting planes which are satisfied as equalities in the last (fractional) LP-solution. These are also called *binding* cuts. This approach is also followed by our B&C algorithm.

4.4.4 Solving the LP-relaxation

Throughout this section we use the standard terminology of Linear Programming, without actually defining every single term. The formal description of technical terms, e.g. primal feasible solution, can be found in any LP textbook (see for example [68, Chapter I.2])

At each node Q^s, $LP(Q^s)$ can be solved by any method for solving linear programs. The standard method is the primal or dual simplex algorithm. Recall that matrix A has significantly more columns than rows. It is usually preferable to have less columns (variables) for the simplex method. In this sense, the dual simplex appears as the most favourable option, although this has to be supported by computational results. Therefore, both the primal and the dual simplex algorithms are tested in Section 4.6. *Interior-point* or *Newton-Barrier* methods have been recently shown to be much more efficient, especially for large LPs ([91]). They are also embedded in commercial solvers ([31]), a fact that allows us to test them as well.

Once $LP(Q^s)$ is solved to optimality, violated inequalities are identified. After adding violated inequalities, the previously optimal solution becomes primal infeasible. The dual solution, however, remains feasible and can be used as the initial solution for the dual simplex algorithm. Hence, it is natural to re-optimise $LP(Q^s)$ via the dual simplex algorithm. In contrast, the primal simplex and the Newton-Barrier method have to reconstruct a primal feasible solution from scratch. Actually, re-optimising with the dual simplex is the default setting in all commercial codes. There exists another, similar, advantage when using dual simplex for initially solving $LP(Q^s)$. Each node has the same matrix rows as its father, unless a row corresponding to a basic variable of the dual solution is deleted during preprocessing. If this is not the case, the optimal solution of the father remains dual feasible in the child. Hence, it may be used again as the initial solution when solving $LP(Q^s)$ with dual simplex.

Our last note concerns the form of the objective function. Since we deal with a feasibility problem. the actual objective function is of no real importance. Notice that the original objective function of $(IP1)$ has a coefficient of 1 for every variable. This implies that the right-hand side of all dual constraints will be identical, thus allowing the appearance of degeneracy. An objective function with different coefficients for each variable could easily resolve this problem. Therefore, the coefficients of the objective function used when solving LPs are randomly generated. Any integer in the interval $[1, n^4]$ is randomly allocated to a single variable.

4.5 Implementation of the Branch & Cut algorithm

The B&C algorithm has been implemented by using the commercial package XPRESS-MP ([31]). Apart from being a mixed integer programming solver, XPRESS-MP provides full flexibility in terms of allowing the user to intervene in various aspects of the solver. This is possible mainly via the Optimiser Subroutine Library XOSL. This library contains all the parameters controlling the solver, along with numerous subroutines, which can be called from a main computer program. Complete documentation of these subroutines is provided in [31]. The main program, including the procedures for preprocessing, branching and cut separation, has been written in the Microsoft C++ programming environment.

XPRESS-MP requires an IP model to be written in an algebraic language, which is essentially $(IP1)$, except for differences in the syntax. The output of the XPRESS modeller is a model in the standard MPS format. Once this MPS format is created, the model can be solved as either an LP or an IP. In the latter case, the Branch and Bound solver is invoked. To clarify matters, we briefly present certain technical details concerning the functionality of the solver.

4.5.1 Matrix operations

The constraint matrix is fully controllable by the user. This aspect allows all characteristics to be read and possibly edited. For example, the coefficients of the objective function can be replaced by randomly generated ones.

The preprocessing mechanism can identify the bounds of the $0-1$ variables and subsequently alter them in order to fix variables to 0 or 1. It may also delete rows and columns of the matrix at a particular node Q^s, without affecting the matrix used in other nodes. Similarly, the branching mechanism can set variables to 0 in order to define new subproblems.

Moreover, the user is allowed to intervene with the simplex algorithm by defining the basis or the pivots. The simplex tableau is available at each point. Once $LP(Q^s)$ is optimised, the simplex tableau is used for detecting the fractional variables and constructing the Gomory fractional cuts. Any cutting plane can be added to the matrix in the form of additional rows.

4.5.2 Control parameters

Numerous control parameters can be handled in order to tune the behaviour of the solver. The first parameter is the algorithm used for solving $LP(Q^s)$, the options being the primal or dual simplex and the Newton-Barrier. The user can switch to any of those algorithms at any point. For example, the initial subproblem can be solved by primal simplex and every linear program thereafter can be solved exclusively by the dual simplex.

XPRESS-MP provides its own preprocessing routine, which can be controlled to perform reduced cost fixing, logical preprocessing, probing or any combination of those three methods (see [78] for details). The solver also incorporates general purpose cutting planes which can be either used or not. Also, node selection can follow various strategies, e.g. depth-first. Finally, technical aspects of numerical precision, like zero tolerance, can be manipulated.

4.5.3 "Callback" subroutines

Perhaps the main strength of XPRESS-MP lies in the set of routines, which can be called at various points of the solution process in order for the user to specify the B&C algorithm. The relatively complex branching mechanism for the OLS problem is implementable only via these routines. These functions are named *callbacks*, exactly because they can be called at each node. A list of all nodes that have not been pruned is maintained by XPRESS-MP. Any of these nodes can be accessed at any point.

At each node of the tree, we are allowed to preprocess and possibly prune the subproblem, before solving $LP(Q^s)$. If the subproblem has to be partitioned, the user can define the bounds applied in the two descendant nodes, the next subproblem to be solved and the information printed as output. The node can be pruned, if infeasible, and the entire solver can terminate if an integer solution is found.

4.5.4 Cut management

A designated set of callback routines handles the cutting planes. As noted above, cutting planes can be simply added as rows of the matrix at each Q^s. However, since certain cuts added at a node are retained at descendant nodes, the same cuts will appear in multiple matrices. For large problem instances, this process occupies unnecessary memory for storing duplicate information. XPRESS-MP overcomes this drawback by maintaining a pool of cutting planes. Hence, additional constraints are no longer added at matrix rows, but explicitly as cuts. Cuts are retained at the "cut pool" only if they have been used in nodes that have not been pruned. Each cutting plane is accompanied by a list of the nodes it appears at. Before a node is solved, all the relevant cuts are added to its matrix. Note that this process takes place automatically, without the user having to intervene.

Specifically, there exists routines for adding cuts even before the original problem is solved. At each node a cut-handling routine is called. Within this routine, the user-provided code can identify violated cuts, add them to the cut pool or to the matrix, and indicate when the iterative procedure of Algorithm 4.1 will terminate. The user can also specify a parameter *delta*, controlling the minimum violation necessary for a cut to be added. Hence, only cuts violated by more than *delta* are considered. This aspect is related to issues of numerical precision. Typical values for *delta* lie in the interval $[10^{-4}, 10^{-2}]$. Another routine is used to identify the cuts to be deleted from both the matrix and the cut pool, i.e. the non-binding cuts.

4.6 Computational analysis

The discussion of the previous section indicates that the B&C algorithm consists of a number of components and each of those includes various parameters. The exact form of the algorithm is difficult to be devised unless we experiment with these parameters. This is exactly the aim of this section. The possible versions of B&C will be applied to the OLS problem for $n = 4,...,12$. This range of values provides a representative sample of the problem. Observe that the $(IP1)$ has 256 variables for $n = 4$ and 20736 variables for $n = 12$. Similarly, the number of equality constraints ranges from 96 ($n = 4$) to 864 ($n = 12$). Orders of 2 and 3 are trivial to handle, whereas orders larger than 12 require considerable computational time. Actually, the exponential explosion in computation time appears at around $n = 10$.

The components of the algorithm examined are the LP-algorithms, the branching rule, the preprocessing procedure, the quality of the cutting planes and cut strategies concerning the number of iterations at each node and the frequency of cut additions. The main performance indicators are the elapsed time and the number of nodes created in the tree. The elapsed time includes time for preprocessing, cut addition and LP-solving (including re-optimisation) and is always measured in seconds. All results were obtained on a Pentium III, 800 MHz, computer with 512Mb of RAM.

4.6.1 Methods for solving LPs

This experiment examines the three possible methods for solving the LP-relaxation of the original subproblem. Hence, only a single LP is solved. The variable fixing indicated by Table 4.2 is implemented in advance. No preprocessing is applied and no cutting planes are incorporated. Table 4.10 illustrates the time (in seconds) required to solve each instance to optimality, having as an objective function the sum of all the variables. It is clear that the Newton-Barrier method becomes significantly faster as the order increases. Especially for $n \geq 9$, the Newton-Barrier method requires at most half the minimum time required by the dual simplex. There is also a slight improvement when using the dual instead of the primal simplex. The only reasonable explanation could be that the dual model has considerably less variables. However, such a small difference may also be caused by a better implementation of dual simplex in this particular version of XPRESS-MP.

Table 4.10: Results for LP-solving algorithms

n	TIME		
	Primal Simplex	Dual Simplex	Newton-Barrier
5	0.51	0.43	1.63
6	1.13	1.04	2.27
7	3.48	3.23	3.21
8	6.72	5.97	4.76
9	10.34	9.41	6.93
10	18.57	16.98	9.19
11	29.24	28.10	13.26
12	35.42	34.71	17.78

4.6.2 Branching rules & objective functions

Three branching rules are examined in this section. The first is variable dichotomy, which is the default option in XPRESS-MP. The rest are the two GUB dichotomy strategies described in Section 4.4.2. For simplicity, we call these branching rules *V_BR* (variable dichotomy), *R_BR* (fixing rows) and *T_BR* (fixing transversals). The algorithm is a Branch & Bound scheme, without any preprocessing. The LP in the top node is solved by the Newton-Barrier method and the dual simplex is applied thereafter. The results in terms of elapsed time and nodes created are depicted in Table 4.11. An upper bound of 5×10^4 seconds has been imposed on the solution time. An asterisk (*) indicates that a certain variant of the algorithm did not manage to find a solution within this time limit. In order to allow for large numbers, the notation $aE + b$ is used, where $a \in [1, 10)$ and $b \in \mathbb{Z}_+$ denotes the power of 10. For example, the number 123456 is written as $1.23E + 5$.

Table 4.11: Results for the branching strategies

n	Time			Nodes		
	V_BR	R_BR	T_BR	V_BR	R_BR	T_BR
4	7	5	4	1	1	1
5	18	13	10	5	3	3
6	1963	1623	1536	20532	16423	14769
7	7991	6843	6437	22037	18978	17524
8	356	292	273	1081	781	688
9	33262	28763	27936	3.71E+5	2.58E+5	2.04E+5
10	38347	33149	32143	5.67E+6	4.05E+6	3.80E+6
11	46934	39847	37852	1.53E+7	9.89E+6	9.33E+6
12	*	*	48174	*	*	7.92E+7

It can be seen that both GUB dichotomy rules create much fewer nodes than variable dichotomy. The creation of fewer subproblems is the reason explaining the reduction in solution time. Method T_BR is clearly better than R_BR, as anticipated by the analysis of Section 4.4.2. Therefore, this is the branching rule applied hereafter. It is worthwhile to notice that even this simple problem specific knowledge is sufficient to drastically improve on a general purpose Branch & Bound scheme. Observe also that the solution time is generally increasing with respect to n, except for $n = 8$. This will be observed in all experiments. The only reason we may suggest is that the first feasible solution, specifically for $n = 8$, appears rather early during the search. Since this is the easiest instance, it is expected that any improvements over simple Branch & Bound will be minimal.

One further experiment is conducted in order to justify the use of a random objective function. We compare algorithm T_BR using two objective functions: the sum of all the variables and a randomly generated one. The results illustrated in Table 4.12 show that a random objective function has absolutely no effect on the number of nodes, but indeed reduces the solution time of LP-relaxations by the dual simplex algorithm, especially as n grows larger.

Table 4.12: The impact of a random objective function

	4	5	6	7	8	9	10	11	12
Time									
Identical coeff.	4	10	1536	6437	273	27936	32143	37852	48174
Random coeff.	4	10	1392	6162	269	27351	31678	37219	47654
Nodes									
Identical coeff.	1	3	14769	17524	688	2.04E+5	3.80E+6	9.33E+6	7.92E+7
Random coeff.	1	3	14769	17524	688	2.04E+5	3.80E+6	9.33E+6	7.92E+7

4.6.3 Preprocessing

Let us now examine the impact of preprocessing on the performance of Branch & Bound (algorithm T_BR). All schemes examined incorporate the branching rule T_BR and a random objective function. We compare the simple B&B algorithm with two more schemes involving also preprocessing at each node. The first uses the embedded preprocessor of XPRESS-MP (algorithm *XPR*). The second applies Algorithm 4.2, where up to n iterations are allowed (algorithm *PREP*). Table 4.13 summarises the results. Results for algorithm T_BR are reproduced from Table 4.12. The fields "Prep." and "LP" contain the percentage of elapsed time employed for preprocessing and LP-solving, respectively. Notice that these percentages do not add exactly to 1. The reason is the extra time spent for setting up a node and "book-keeping" tasks. Field "Nodes", for algorithms XPR and PREP, counts only the nodes where the LP had to be solved, i.e. it does not include nodes pruned by preprocessing alone.

Table 4.13: Results for preprocessing

n	TIME							NODES		
	T_BR	XPR			PREP			T_BR	XPR	PREP
		LP	Prep.	Total	LP	Prep.	Total			
4	4	97.1	0.5	4	96.4	1.2	4	1		1
5	10	97.3	0.5	10	97.1	1.4	10	3		3
6	1392	97.4	0.4	1357	97.4	1.6	1133	14769	14313	12281
7	6162	98.1	0.5	6102	97.3	1.8	5288	13769	13472	11735
8	269	97.3	0.6	267	96.7	1.8	255	688	673	571
9	27351	97.4	0.4	26894	95.9	2.2	24373	2.04E+5	2.03E+5	1.94E+5
10	31678	98.5	0.4	30672	96.7	2.1	28537	3.80E+6	3.78E+6	3.48E+6
11	37219	97.4	0.5	36149	96.6	2.7	34640	9.33E+6	9.32E+6	8.92E+6
12	47654	98.2	0.4	47019	97.1	2.2	44109	7.92E+7	7.89E+7	7.51E+7

Table 4.13 shows that the preprocessing routine of XPRESS-MP achieves insubstantially better results in terms of both nodes and time, compared to simple Branch & Bound. Algorithm 4.2, in contrast, improves considerably the results with respect to elapsed time. Although the number of nodes is not much smaller, each node is easier to solve, particularly at lower levels of the tree. This occurs exactly because of the drastic decrease in the matrix size. A limited proportion of subproblems is identified as infeasible before even solving the LP, basically because the left-hand side of a constraint becomes empty. It can be finally concluded that the preprocessing effort for algorithm PREP is negligible, compared to the benefits produced.

4.6.4 Quality of cutting planes

As discussed in Section 4.4.3, the quality of cutting planes refers to their ability to prune an infeasible subproblem without further branching and to the computational effort required to generate them. Under this perspective, we must experiment with the order, according to which families of valid inequalities are examined and added to the matrix. This is mandatory, despite the fact that algorithmic correctness indicates that cliques should be examined first. Nevertheless, the main finding of our experimentation is that clique inequalities are indeed the most valuable, therefore they should be the first to be examined.

The following experiment uses algorithm PREP of the previous section. Recall that this algorithm performs preprocessing, branches in a way that fixes transversals and applies a random objective function. The additional element is that a single family of cutting planes is separated at every node where at least one variable is set to 1. Violated inequalities are added for up to n iterations and at most $2 \cdot n$ cuts are added at each iteration. Overall, at most $2n^2$ cuts can be added at a particular node. The only exception to this rule is the top node of tree, i.e. the original problem, where at most n^2 cuts are added for up to $2 \cdot n$ iterations.

These upper bounds have been selected only for simplicity of illustration. Experiments for different values of these upper bounds have been conducted and have provided analogous comparative results. As noted earlier, only binding cuts are retained in descendant nodes. Four different variants are tested, each for one of clique, lifted 5-hole, lifted antiweb and (fractional) Gomory inequalities. Results concerning the elapsed time are illustrated in Table 4.14. Column "NO CUTS" replicates column "PREP" of Table 4.13. No results for the preprocessing time are illustrated, since they are matching the ones of Table 4.13. Fields "LP" and "Cuts" contain the percentage of elapsed time employed for LP-solving and cut generation, respectively.

Table 4.14: Branch & Cut with a single family of cutting planes

n	TIME												
	NO CUTS	CLIQUE			5-HOLE			ANTIWEB			GOMORY		
		LP	Cuts	Total	LP	Cuts	Total	LP	Cuts	Total	LP	Cuts	Total
4	4	98.3	0	4	98.4	0	4	98.1	0	4	98.2	0	4
5	10	95.2	3.2	12	91.6	6.3	14	92.8	4.7	12	95.4	1.6	16
6	1133	94.8	3.0	634	92.3	5.1	965	92.6	5.1	1012	96.7	1.4	1273
7	5288	94.4	3.1	3847	91.5	6.2	4784	93.1	5.7	4972	96.1	2.1	5304
8	255	95.6	2.8	252	92.4	6.1	264	92.7	5.6	257	94.4	2.1	331
9	24373	95.2	2.9	19169	92.9	5.4	22417	92.3	5.2	23102	96.2	1.8	25027
10	28537	95.2	2.8	23295	92.5	5.7	26851	92.4	5.6	27346	95.3	1.7	29351
11	34640	94.8	3.3	25907	93.1	5.8	30287	92.9	5.2	29187	96.5	1.8	36416
12	44109	94.6	3.3	33138	91.9	5.6	39574	93.0	4.9	37294	95.8	1.8	47381

According to Table 4.14, the best results are achieved by clique inequalities. Incorporating these inequalities results in a striking reduction of elapsed time compared to B&B. Moreover, the computational time required for their identification appears to be negligible. Gomory inequalities are proved to be the weakest ones but the cheapest to generate, as shown by the percentage of time devoted to identify them. Actually, using Gomory inequalities gives worse results than simple B&B, exactly because of the re-optimisation tasks at each node. Lifted 5-hole inequalities are the most expensive to generate, although they achieve the best results after clique inequalities, for orders up to 10. For $n = 11, 12$, antiweb inequalities behave more efficiently than 5-hole inequalities. In general, generation of antiwebs requires more effort than separation of cliques but less than separation of 5-holes.

The results of Tables 4.15 and 4.16 complement these observations. This table depicts results related to the number of nodes, and the average number of cuts added at nodes, where cut addition took place. Gomory inequalities are added in large numbers, close to the maximum allowed at each node. This happens because one can always trace enough fractional variables at each LP-solution and generate a Gomory fractional cut for each one of them. However, adding these cuts to the matrix does not produce better results in terms of nodes; the same subproblems are examined and repeatedly optimised with additional constraints, thus spending extra time, without pruning any further nodes. It is therefore this reason that allows B&B (without cuts) to be better in terms of time. The beneficial contribution of cliques becomes more apparent, since Table 4.15 demonstrates that they also create the smallest number of nodes. Hence, clique inequalities are the best in terms of identifying infeasible subproblems in the higher possible tree levels. This is no surprise since, as reported in [6], clique inequalities are the most useful regarding general set-partitioning problems. Lifted 5-hole inequalities are quite strong in terms of proving infeasibility because they consistently create slightly more nodes than cliques. The fact that a large number of those is added at each node explains the excess computational time needed for their generation. Finally, antiweb inequalities prove less useful, since only a small fraction of those is violated, i.e. much fewer are added at each node. The reduction they achieve in the number of nodes is also not remarkable.

In conclusion, clique inequalities should indeed by the first ones to be examined and added to the matrix. Once no more violated clique inequalities can be found, separation of odd-hole and antiweb inequalities can proceed. Gomory inequalities are to be checked only if no other family can provide a cutting plane.

Table 4.15: Number of nodes when a single family of cutting planes is used

n	NODES				
	NO CUTS	CLIQUE	5-HOLE	ANTIWEB	GOMORY
4	1	1	1	1	1
5	3	1	1	1	3
6	12281	6871	7462	9649	12273
7	11735	3593	5076	7861	11735
8	571	178	184	407	571
9	1.94E+5	58123	61773	92136	1.94E+5
10	3.48E+6	7.31E+5	7.44E+5	9.31E+5	3.48E+6
11	8.92E+6	1.27E+6	1.35E+6	3.47E+6	8.92E+6
12	7.51E+7	4.23E+6	4.58E+6	6.23E+6	7.51E+7

Table 4.16: Cuts added per node

n	CUTS PER NODE				
	CLIQUE	5-HOLE	ANTIWEB	GOMORY	MAX ($2n^2$)
4	0	0	0	0	32
5	25	27	29	49	50
6	46	62	32	70	78
7	59	78	39	95	98
8	67	86	56	123	128
9	83	105	68	162	162
10	129	147	104	197	200
11	152	193	121	240	242
12	177	214	163	288	288

In order to verify this argument, we have also experimented with different orders for examining the various families. These variants of B&C incorporate separation algorithms for all families, each with a different priority. For example, we have tried to add antiweb inequalities first and examine all others only if no antiweb is violated. The results have been worse than the ones provided by using only clique inequalities. Since clique and lifted 5-hole inequalities are the strongest, the sensible options are to separate first one of these two. Algorithm "Cliques first" separates clique inequalities first and odd-hole and antiwebs afterwards, i.e. when no more cliques are violated. In contrast, algorithm "Odd-holes first" focuses first on separating lifted 5-hole inequalities. Gomory cuts are not considered at all. The results of these experiments are presented in Table 4.17.

Table 4.17: Branch & Cut with different orders of separation

n	TIME							NODES	
	CLIQUES FIRST			ODD-HOLES FIRST				CLIQUES FIRST	ODD-HOLES FIRST
	LP	Cuts	Total	LP	Cuts	Total			
4	98.3	0	4	98.3	0	4		1	1
5	94.8	3.7	9	91.3	6.3	14		1	1
6	94.3	4.1	553	92.5	5.1	975		3167	7127
7	94.5	3.7	2325	91.5	6.2	4735		280	5061
8	95.2	3.2	252	92.4	6.1	268		57	184
9	94.8	3.5	17854	92.9	5.4	22371		33102	54792
10	94.9	3.3	21561	92.5	5.7	26792		5.11E+5	6.834E+5
11	94.7	3.6	24813	93.1	5.8	29858		9.12E+5	1.02E+6
12	94.8	3.6	31642	91.9	5.6	39611		2.74E+6	4.22E+6

Algorithm "Odd-holes first" remains consistently worse in terms of both elapsed time and nodes created. Its performance resembles the one of the B&C algorithm which uses 5-holes exclusively (Tables 4.14 and 4.15). The small improvement achieved is justified by the parallel use of cliques.

The "Cliques first" scheme exhibits significant progress, even when compared with the algorithm that exclusively uses clique inequalities. For example, the instance of $n = 7$ is solved in only 280 nodes, compared to the 11,735 nodes of B&B and the 3,593 nodes of B&C with cliques only. This occurs because the combined performance of cliques and 5-holes forces the LP to select an integral extreme point, at this very early point. The same reason explains the reduction in the number of nodes for $n = 8$. For the infeasible case of $n = 6$, the entire solution space is searched in only 3,167 nodes, compared to 12,281 nodes for B&B and 6,871 nodes for B&C with cliques only. An important observation, not presented in Table 4.17, is that, once no violated clique can be found, violated antiwebs are rarely identified. In contrast, separation of lifted 5-hole inequalities is able to provide cuts for several more iterations. Another critical aspect is that the maximum number of iterations is seldom reached and only at higher tree levels; usually, no violated cut can be found after 4 or 5 iterations.

Given these final results, the "Cliques first" variant is retained as the standard one hereafter.

4.6.5 Cut strategies

This section examines mechanisms controlling the aggressiveness of cut addition. The features of the code related to this aspect are the maximum number of cuts added at each iteration, the maximum number of iterations and, most important, the fraction of subproblems eligible for cut addition. There can exist multiple choices and heuristic criteria for each of these parameters. Nevertheless, the requirement of a reasonably long polynomial procedure at each node implies that both the maximum number of cuts and the maximum number of iteration can only be a multiple of n or n^2. Once more, we only report on three representative versions, named "conservative", "moderate" and "aggressive" cut strategies.

The conservative approach, adds only n cuts for up to n iterations at a certain node. For clique inequalities, it is implied that up to $\frac{n}{2}$ violated cuts are added from each class. Eligible nodes are the ones where a variable is set to 1, and up to the first $f(n)$ levels, where $f(n) = \lceil [(n-3) + 2(n-1)] \, log_2 n \rceil$. This limit implies examining nodes only up to the point where the first column and at least a single transversal have been fixed. Deeper in the tree, the algorithm performs only preprocessing. The aggressive approach adds up to n^2 violated cuts for up to $2 \cdot n$ iterations. Cut are added whenever a variable is set to 1, or at most every 2 levels of the tree. The moderate approach selects intermediate options. The values of all parameters are illustrated in Table 4.18. Notice that the maximum number of cuts and the maximum number of iterations is always larger in the top node.

Table 4.19 summarises the results obtained. It is easy to see that the moderate approach performs better and almost identically to the "Cliques first" scheme of Table 4.17. It performs slightly better for small orders and worse for $n \geq 10$. Notice that the moderate cut strategy is still more aggressive than the cut strategy of the original "Cliques first" algorithm. Hence, our first conclusion is that it is better to insist on cut addition for small orders and become more moderate for larger orders. In this context, observe that the conservative approach spends less time for cut separation, creates constantly more nodes and becomes better than the moderate approach for $n \geq 11$. To the contrary, the aggressive strategy creates substantially fewer subproblems but is computationally more expensive, since it requires considerable effort at each node. Actually, it exceeds the time limit for $n = 12$. These results reflect a natural trade-off of Branch & Cut algorithms between the number of nodes and the time consumed in each one. Notice that an aggressive approach increases also the time for re-optimising LP-relaxations, since it adds a large number of cuts at each iteration. It must be emphasised, however, that the ability of early inferring that a certain branch leads to infeasibility, is strongly correlated to the aggressiveness of the cut strategy. The actual equilibrium is certainly problem-specific, and probably instance-specific as well.

Table 4.18: Strategies for cut addition

	CONSERVATIVE	MODERATE	AGGRESSIVE
maximum cuts (top node/ other nodes)	$2n$ / n	n^2 / $\frac{n^2}{2}$	$2n^2$ / n^2
maximum iterations	n	$4n/2n$	$4n/2n$
criterion for node	variable set to 1, for the first $\lceil[2(n-1)+(n-3)]\cdot\log_2 n\rceil$ levels	variable set to 1	variable set to 1 or every 2 levels

Table 4.19: Results for three cut strategies

n	TIME									NODES		
	Conservative			Moderate			Aggressive			Cons.	Mod.	Aggr.
	LP	Cuts	Total	LP	Cuts	Total	LP	Cuts	Total			
4	98.6	0	4	98.4	0	4	98.4	0	4	1	1	1
5	96.5	2.6	11	94.7	3.3	11	94.8	3.6	12	1	1	1
6	97.4	1.8	892	94.4	4.2	527	93.4	4.7	1851	3243	2861	2159
7	96.5	1.9	3870	94.6	3.7	3278	93.6	4.9	4153	316	274	185
8	97.4	1.6	247	95.1	3.1	254	94.1	4.8	482	61	57	49
9	97.7	1.2	19639	95.2	3.0	17592	92.9	5.3	24176	35726	32542	28735
10	97.9	1.4	24816	94.9	3.3	21903	93.5	4.6	29749	5.42E+5	3.86E+5	2.73E+5
11	97.6	1.2	22531	94.7	3.1	25713	93.2	5.1	32978	8.24E+5	7.71E+5	4.17E+5
12	97.7	1.1	32187	94.9	3.4	33872	*	*	*	4.12E+5	2.37E+6	*

Our last note concerns the case of $n = 6$. Our B&C algorithm provides a computer-based proof of infeasibility via enumeration. However, it would be desirable to obtain a polyhedral proof, analogous to that of Section 4.3. This would require solving the problem at a single node, thus providing a linear system which proves infeasibility. Unfortunately, this has not been possible, even with an aggressive cut strategy that incorporates Gomory fractional cuts for an enormous number of iterations, e.g. 1000. This illustrates already the limitation of this approach or the lack of sufficient polyhedral knowledge for the OLS problem. It is still worthwhile to mention that the B&C algorithm, with a moderate cut strategy, proves infeasibility for $n = 6$ at most after fixing a single transversal.

Table 4.20: Improvements over simple Branch & Bound

n	TIME				
	BRANCH & BOUND	GUB BRANCHING	RANDOM O.F.	PREPROCESSING	CUTTING PLANES
4	7	4	4	4	4
5	18	10	10	10	11
6	1963	1536	1392	1133	527
7	7991	6437	6162	5288	3278
8	356	273	269	255	254
9	33262	27936	27351	24373	17592
10	38347	32143	30678	28537	21903
11	46934	37852	37219	34640	25713
12	*	48174	47654	44109	33872

4.7 Concluding remarks

In this chapter, we have reported on computational work, related to the application of IP to solve the OLS problem. For the infeasible case of $n = 6$, we have proposed an alternative, LP-based, proof. The proof is accomplished only after preliminary variable fixing, which arises from a certain method for reducing problem's symmetry. We have also presented a Branch & Cut algorithm for the OLS problem. This algorithm uses the polyhedral developments of the previous chapters and exploits the structure of the problem to devise efficient preprocessing and branching mechanisms. The actual form of the algorithm is determined via experimenting with several parameters. Computational results show that the algorithm improves significantly over a commercial solver and that strong valid inequalities constitute, apart from a theoretical description, a powerful solution tool. As a summary of all previous results, Table 4.20 shows the performance of B&B with variable dichotomy and the improved performance after incorporating, one by one, the branching mechanism, the randomised objective function, the preprocessing procedure and the moderate cut strategy of the previous section. This table shows that the most drastic reduction in solution time appears after applying the cutting planes.

On the other hand, our results show that at least 95% of computation time is devoted to solving and re-optimising linear programming relaxations. Hence, limitations on the B&C approach are imposed by a large matrix size, at least in the higher tree levels, along with the absence of faster re-optimisation methods. Next chapter introduces an enumeration paradigm, whose performance is not restricted by the necessity to solve linear programs.

Chapter 5

Constraint Programming algorithms

This chapter presents the application of Constraint Programming (CP) to the OLS problem. Its first part serves the purposes of a review of the CP approach. Hence, Section 5.1 defines Constraint Satisfaction Problems (CSP) and presents the notions of constraint graph and arc-consistency. Examples involving Latin squares are used to illustrate these concepts. Methods developed to solve CSPs are presented in Section 5.2, where their performance is also analysed. There exists a broad spectrum of such methods, from the naive Generate & Test scheme to more sophisticated ones, like Conflict-Directed Backjumping. The critical aspects concerning the design of a CP algorithm are extensively discussed. The actual application of CP methods to the OLS problem is exhibited in Section 5.3. Several branching strategies and solution schemes, which exploit the problem structure in different ways, are proposed. Computational results are presented in Section 5.4.

5.1 The Constraint Programming paradigm

Constraint logic programming or *Constraint Programming* (CP) is a method for solving a large variety of problems. The origins of the method are located within the Artificial Intelligence (AI) community. Its main area of application has been feasibility problems arising from the computer science literature, although recent developments ([40]) suggest that CP could serve as a method for general combinatorial optimisation problems. Examples include machine vision, belief maintenance and scheduling ([57]), along with the generic satisfiability problem (SAT). The latter is defined on a number of logical sentences on "$true - false$" (binary) variables, and asks for an assignment that makes all sentences simultaneously *true*. The research field focusing on SAT problems is referred to as *Logic Programming*. In a sense, Constraint Logic Programming can be regarded as the extension of Logic Programming to problems involving variables with domains of cardinality greater than 2. Before presenting a number of formal definitions of CP, we apply Logic Programming to the OLS problem for $n = 2$.

5.1.1 Logic Programming applied to OLS for $n = 2$

This section illustrates a proof of the non-existence of a pair of OLS for $n = 2$. It acts as an example for the use of logic methods for inference. We first formulate the OLS problem for $n = 2$ in the form of logical sentences. Assume Latin squares X and Y and let variables $x_{ij}, y_{ij} \in \{0,1\}$, $i,j = 0,1$, denote the value contained in cell (i,j) of square X and Y, respectively.

We briefly review the elementary concepts of Logic Programming. An *atomic proposition* is simply a binary variable, e.g. x_{00}. A *literal* is an atomic proposition or its negation, e.g. x_{00} or $\neg x_{00}$. A *formula* or *proposition* is a sentence of atomic propositions, using the symbols $\{\vee, \wedge\}$. Any subformula which is a pure disjunction of literals, i.e. uses only operator $\vee$, is called a *clause*. It is common to express a formula as a conjunction of clauses or, formally, in *conjunctive normal form* (CNF). Finally, a *model* is an assignment of values to literals, such that all clauses are simultaneously satisfied.

There exists an alternative representation of formulas in the form of rules. For example, if pair $(0,0)$ appears in cell $(0,0)$, both variables x_{00} and y_{00} must be 0, i.e. variables $\neg x_{00}$ and $\neg y_{00}$ must be 1. As a consequence, pair $(0,0)$ cannot appear in position $(0,1)$, therefore variables x_{01} and y_{01} should be 1. The rule for expressing this fact is

$$\neg x_{00} \wedge \neg y_{00} \rightarrow x_{01} \vee y_{01} \tag{5.1}$$

where symbol '$\rightarrow$' stands for "implies", x_{00}, y_{00} are the *antecedents* and x_{01}, y_{01} form the *consequent* of the rule ([23, Section 2.1.3]). It is easy to see that rule (5.1) is equivalent to the clause:

$$(x_{00} \vee y_{00} \vee x_{01} \vee y_{01})$$

All rules ensuring that, if the pair $(0,0)$ appear at the cells $(0,0)$, it should not appear in any other cell, are:

$$\begin{aligned} \neg x_{00} \wedge \neg y_{00} &\rightarrow x_{01} \vee y_{01} \\ \neg x_{00} \wedge \neg y_{00} &\rightarrow x_{10} \vee y_{10} \\ \neg x_{00} \wedge \neg y_{00} &\rightarrow x_{11} \vee y_{11} \end{aligned} \tag{5.2}$$

Rules (5.2) are equivalent to the clauses:

$$\begin{aligned} &(x_{00} \vee y_{00} \vee x_{01} \vee y_{01}) \\ &(x_{00} \vee y_{00} \vee x_{10} \vee y_{10}) \\ &(x_{00} \vee y_{00} \vee x_{11} \vee y_{11}) \end{aligned} \tag{5.3}$$

Three more sets of clauses can be constructed by assuming that pair $(0,0)$ appears in

cell $(0,1)$:

$$\begin{aligned} &(x_{01} \vee y_{01} \vee x_{00} \vee y_{00}) \\ &(x_{01} \vee y_{01} \vee x_{10} \vee y_{10}) \\ &(x_{01} \vee y_{01} \vee x_{11} \vee y_{11}) \end{aligned} \tag{5.4}$$

cell $(1,0)$:

$$\begin{gathered}(x_{10} \vee y_{10} \vee x_{00} \vee y_{00}) \\ (x_{10} \vee y_{10} \vee x_{01} \vee y_{01}) \\ (x_{10} \vee y_{10} \vee x_{11} \vee y_{11})\end{gathered} \tag{5.5}$$

cell $(1,1)$:

$$\begin{gathered}(x_{11} \vee y_{11} \vee x_{00} \vee y_{00}) \\ (x_{11} \vee y_{11} \vee x_{01} \vee y_{01}) \\ (x_{11} \vee y_{11} \vee x_{10} \vee y_{10})\end{gathered} \tag{5.6}$$

After removing any duplications, the set of clauses enforcing that pair $(0,0)$ must occur exactly once are:

$$\begin{gathered}(x_{00} \vee y_{00} \vee x_{01} \vee y_{01}) \\ (x_{00} \vee y_{00} \vee x_{10} \vee y_{10}) \\ (x_{00} \vee y_{00} \vee x_{11} \vee y_{11}) \\ (x_{01} \vee y_{01} \vee x_{10} \vee y_{10}) \\ (x_{01} \vee y_{01} \vee x_{11} \vee y_{11}) \\ (x_{10} \vee y_{10} \vee x_{11} \vee y_{11})\end{gathered} \tag{5.7}$$

Three more sets of 6 clauses each represent the fact that each of the remaining three pairs must also appear exactly once.

For pair $(0,1)$ the constraints are:

$$\begin{gathered}(x_{00} \vee \neg y_{00} \vee x_{01} \vee \neg y_{01}) \\ (x_{00} \vee \neg y_{00} \vee x_{10} \vee \neg y_{10}) \\ (x_{00} \vee \neg y_{00} \vee x_{11} \vee \neg y_{11}) \\ (x_{01} \vee \neg y_{01} \vee x_{10} \vee \neg y_{10}) \\ (x_{01} \vee \neg y_{01} \vee x_{11} \vee \neg y_{11}) \\ (x_{10} \vee \neg y_{10} \vee x_{11} \vee \neg y_{11})\end{gathered} \tag{5.8}$$

For pair $(1,0)$ the constraints are:

$$\begin{gathered}(\neg x_{00} \vee y_{00} \vee \neg x_{01} \vee y_{01}) \\ (\neg x_{00} \vee y_{00} \vee \neg x_{10} \vee y_{10}) \\ (\neg x_{00} \vee y_{00} \vee \neg x_{11} \vee y_{11}) \\ (\neg x_{01} \vee y_{01} \vee \neg x_{10} \vee y_{10}) \\ (\neg x_{01} \vee y_{01} \vee \neg x_{11} \vee y_{11}) \\ (\neg x_{10} \vee y_{10} \vee \neg x_{11} \vee y_{11})\end{gathered} \tag{5.9}$$

For pair $(1,1)$ the constraints are:

$$\begin{aligned}
&(\neg x_{00} \vee \neg y_{00} \vee \neg x_{01} \vee \neg y_{01})\\
&(\neg x_{00} \vee \neg y_{00} \vee \neg x_{10} \vee \neg y_{10})\\
&(\neg x_{00} \vee \neg y_{00} \vee \neg x_{11} \vee \neg y_{11})\\
&(\neg x_{01} \vee \neg y_{01} \vee \neg x_{10} \vee \neg y_{10})\\
&(\neg x_{01} \vee \neg y_{01} \vee \neg x_{11} \vee \neg y_{11})\\
&(\neg x_{10} \vee \neg y_{10} \vee \neg x_{11} \vee \neg y_{11})
\end{aligned} \tag{5.10}$$

Additional clauses are required to model the fact that each square is Latin. For example, if x_{00} is 0, both x_{01} and x_{10} must be 1. The associated rules are:

$$\begin{aligned}
\neg x_{00} &\rightarrow x_{01}\\
\neg x_{00} &\rightarrow x_{10}
\end{aligned}$$

These are equivalent to the clauses:

$$\begin{aligned}
&x_{00} \vee x_{01}\\
&x_{00} \vee x_{10}
\end{aligned}$$

Repeating the procedure for all cells and both values, and after deleting duplicate clauses, yields the set:

$$\begin{aligned}
&(x_{00} \vee x_{01})\\
&(x_{00} \vee x_{10})\\
&(x_{01} \vee x_{11})\\
&(x_{10} \vee x_{11})\\
&(\neg x_{00} \vee \neg x_{01})\\
&(\neg x_{00} \vee \neg x_{10})\\
&(\neg x_{01} \vee \neg x_{11})\\
&(\neg x_{10} \vee \neg x_{11})
\end{aligned} \tag{5.11}$$

The set of clauses for square Y are analogous. The number of clauses can be reduced by erasing clauses that are dominated. It is said that a clause C_s ***absorbs*** a clause C_t if all literals of C_s appear in C_t. A clause C_t is *implied* by a clause C_s if C_t is a tautology or if C_s absorbs C_t. Observe that the first two and the last two of clauses (5.7)-(5.10) are implied by one of the clauses (5.11) and by one of the corresponding clauses for square Y. For example the clause $(x_{00} \vee y_{00} \vee x_{01} \vee y_{01})$ of (5.7) is absorbed by both clauses $(x_{00} \vee x_{01})$ and $(y_{00} \vee y_{01})$. Table 5.1 illustrates the set of non-implied clauses for the OLS problem for $n = 2$.

Algorithms for solving models in logic (formulas) are based on the *principle of resolution* ([23, Section 2.1.2]):

> In any CNF formula, suppose there are two clauses C_s, C_t with exactly one atomic proposition p appearing negated in C_s and posited in C_t. It is then possible to resolve C_s and

Table 5.1: Formula for OLS of order 2

C_1 :	$(x_{00} \vee y_{00} \vee x_{11} \vee y_{11})$	C_9 :	$(x_{00} \vee x_{01})$	C_{17} :	$(y_{00} \vee y_{01})$
C_2 :	$(x_{01} \vee y_{01} \vee x_{10} \vee y_{10})$	C_{10} :	$(x_{00} \vee x_{10})$	C_{18} :	$(y_{00} \vee y_{10})$
C_3 :	$(x_{00} \vee \neg y_{00} \vee x_{11} \vee \neg y_{11})$	C_{11} :	$(x_{01} \vee x_{11})$	C_{19} :	$(y_{01} \vee y_{11})$
C_4 :	$(x_{01} \vee \neg y_{01} \vee x_{10} \vee \neg y_{10})$	C_{12} :	$(x_{10} \vee x_{11})$	C_{20} :	$(y_{10} \vee y_{11})$
C_5 :	$(\neg x_{00} \vee y_{00} \vee \neg x_{11} \vee y_{11})$	C_{13} :	$(\neg x_{00} \vee \neg x_{01})$	C_{21} :	$(\neg y_{00} \vee \neg y_{01})$
C_6 :	$(\neg x_{01} \vee y_{01} \vee \neg x_{10} \vee y_{10})$	C_{14} :	$(\neg x_{00} \vee \neg x_{10})$	C_{22} :	$(\neg y_{00} \vee \neg y_{10})$
C_7 :	$(\neg x_{00} \vee \neg y_{00} \vee \neg x_{11} \vee \neg y_{11})$	C_{15} :	$(\neg x_{01} \vee \neg x_{11})$	C_{23} :	$(\neg y_{01} \vee \neg y_{11})$
C_8 :	$(\neg x_{01} \vee \neg y_{01} \vee \neg x_{10} \vee \neg y_{10})$	C_{16} :	$(\neg x_{10} \vee \neg x_{11})$	C_{24} :	$(\neg y_{10} \vee \neg y_{11})$

C_t on x. We call C_s and C_t the *parents* of the *resolvent* clause, which is formed by the disjunctions of literals in C_s and C_t, excluding p and $\neg p$. The resolvent is an *implication* of the parent clauses.

As an example, if we resolve clauses C_1 and C_{13} on x_{00}, we derive the clause:

$$(\neg x_{01} \vee y_{00} \vee x_{11} \vee y_{11})$$

The notation $Res(C_s, C_t, p)$ denotes the resolution of clauses C_s, C_t on atomic proposition p. Recall also that $p \vee p = p$, $p \vee \neg p = 1$. The proof of infeasibility can be conducted in steps. Each step applies resolution to the original clauses plus clauses generated in all previous steps.

Step 1

$Res(C_1, C_{13}, x_{00})$:	$(\neg x_{01} \vee y_{00} \vee x_{11} \vee y_{11})$	C_{25}
$Res(C_3, C_{13}, x_{00})$:	$(\neg x_{01} \vee \neg y_{00} \vee x_{11} \vee \neg y_{11})$	C_{26}
$Res(C_5, C_{11}, x_{11})$:	$(\neg x_{00} \vee y_{00} \vee x_{01} \vee y_{11})$	C_{27}
$Res(C_7, C_{11}, x_{11})$:	$(\neg x_{00} \vee \neg y_{00} \vee x_{01} \vee \neg y_{11})$	C_{28}

Step 2

$Res(C_{21}, C_{25}, y_{00})$:	$(\neg x_{01} \vee \neg y_{01} \vee x_{11} \vee y_{11})$	C_{29}
$Res(C_{17}, C_{26}, y_{00})$:	$(\neg x_{01} \vee y_{01} \vee x_{11} \vee \neg y_{11})$	C_{30}
$Res(C_{21}, C_{27}, y_{00})$:	$(\neg x_{00} \vee \neg y_{01} \vee x_{01} \vee y_{11})$	C_{31}
$Res(C_{17}, C_{28}, y_{00})$:	$(\neg x_{00} \vee y_{01} \vee x_{01} \vee \neg y_{11})$	C_{32}

Step 3

$Res(C_{23}, C_{29}, y_{11})$:	$(\neg x_{01} \vee \neg y_{01} \vee x_{11})$	C_{33}
$Res(C_{19}, C_{30}, y_{11})$:	$(\neg x_{01} \vee y_{01} \vee x_{11})$	C_{34}
$Res(C_{23}, C_{31}, y_{11})$:	$(\neg x_{00} \vee \neg y_{01} \vee x_{01})$	C_{35}
$Res(C_{19}, C_{32}, y_{11})$:	$(\neg x_{00} \vee y_{01} \vee x_{01})$	C_{36}

Step 4

$Res(C_{33}, C_{34}, y_{01})$:	$(\neg x_{01} \vee x_{11})$	C_{37}
$Res(C_{35}, C_{36}, y_{01})$:	$(\neg x_{00} \vee x_{01})$	C_{38}

Step 5

$Res(C_{15}, C_{37}, x_{11})$:	$(\neg x_{01})$	C_{39}
$Res(C_9, C_{38}, x_{00})$:	(x_{01})	C_{40}

Step 6

$Res(C_{39}, C_{40}, x_{01})$:	$()$	

This last empty clause illustrates that there exits no model for the set of clauses of Table 5.1. As a result, there exists no pair of OLS of order 2.

Although it has been trivial to prove the result for this small instance, a systematic algorithm is necessary for generating implied clauses for larger instances. It is possible that the resolution procedure reaches a point where no more resolvents can be produced ([23, Section 2.2.2]). The only solution at this point is to select a variable and create two subproblems, each having this variable set to 0 or 1. This enumerative mechanism is the basis of the Davis-Putnam-Loveland procedure, which remains the standard exact method for solving satisfiability problems (in [32], reproduced in[23, Section 3.1.4]). Note that this is essentially a Branch & Bound method, in terms of IP terminology, and is analogous to the implicit enumeration algorithm of [7].

A formulation for the OLS problem of order n as a set of clauses, can only be based on binary variables x_{ijkl}, exactly as in model $(IP1)$. The reason is that the resolution algorithm requires binary variables. The generalisation of the resolution procedure for variables with domains of cardinality greater than 2, is the k-consistency algorithm ([28]) also called *multivalent resolution* (in [53, Section 4.2]). This is one of the topics addressed in the Constraint Programming literature, the subject of the forthcoming sections.

5.1.2 Definitions

The problems addressed by CP can be formalised according to the following definition of a *Constraint Satisfaction Problem* (CSP).

Definition 5.1 *A CSP consists of:*

- *A finite set of variables* $X = \{x_1, x_2, \ldots, x_m\}$.
- *Finite set* $D_i, i = 1, ..., m$, *of possible values, which can be assigned to a variable* x_i, *i.e. the domain of the variable. The contents of* D_i *are assumed to be integers, although they can generally be of any arbitrary type.*
- *A finite set of constraints restricting the values that the variables can simultaneously receive. Formally, a constraint* $C(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$ *involving the variables* $x_{j_1}, x_{j_2}, \ldots, x_{j_m}$ *can be any subset of the possible combinations of values for* $x_{j_1}, x_{j_2}, \ldots, x_{j_m}$, *i.e.* $C(x_{j_1}, x_{j_2}, \ldots, x_{j_m}) \subseteq D_{j_1} \times D_{j_2} \times ... \times D_{j_m}$. *This subset is the set of all tuples* $(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$ *allowed by the constraint.*

It follows that a CSP can be denoted as a triple CSP(X, D, C). There is no further assumption about the form of each constraint, e.g. linearity. Therefore, the CSP scheme can incorporate a broad class of constraints, including non-linear ones, and express them in a compact way. Definition 5.1 is borrowed from [83], where a more detailed introduction to CP can be found. The absence of an objective function from this definition reveals the emphasis of CP literature on feasibility, rather than optimality, problems.

As an example, consider the variables x_1 and x_2, with $D_1 = \{1,2,3\}$ and $D_2 = \{2,3\}$. Any subset of the tuples $\{(1,2), (1,3), (2,2), (2,3), (3,2), (3,3)\}$ could be a valid constraint on x_1 and x_2. The constraint $x_1 \neq x_2$ is represented by the subset $\{(1,2), (1,3), (2,3), (3,2)\}$, whereas the constraint $x_1 \cdot x_2 < 4$ is represented by the subset $\{(1,2), (1,3)\}$. Since the set of tuples defining a constraint can be absolutely arbitrary, a constraint does not have to be expressible in a closed form, e.g. $x_1 \neq x_2$.

The number of variables included in a constraint is called the *arity* of the constraint and can be any number between 1 and n. A constraint with an arity of 1 is called a *unary* constraint, while one with an arity of 2 is called a *binary* constraint.

Example 5.1 *In the case of OLS, let X & Y denote the two squares. Define one variable for each cell of the two squares, i.e. X_{ij} & Y_{ij}, $i, j = 0, \ldots, n-1$, where indices i and j refers to the rows and columns, respectively. The domain associated with every variable X_{ij} is $D = D_{X_{ij}} = \{0, \ldots, n-1\}$, the domain for each variable Y_{ij} being similarly defined.*

We know that each domain member must appear exactly once in each row and column. Consequently, any two variables, which have at least one index in common, i.e. X_{12} & X_{13}, must be assigned different values. For each row/column, there are $\binom{n}{2} = \frac{n(n-1)}{2}$ binary constraints of the form $X_{12} \neq X_{13}$, one for each pair of variables having one index in common. Since each such set of constraints must be repeated for each row and column, there are $n^2(n-1)$ constraints in total. The same constraints must be imposed on variables Y_{ij}. Additional constraints are necessary in order to assure that the two squares are orthogonal. The value of each $X_{i_0 j_0}$, in conjunction with the value of $Y_{i_0 j_0}$, is constraining the values for all other $X_{i_1 j_1}$'s and $Y_{i_1 j_1}$'s ($i_0 \neq i_1$ or $j_0 \neq j_1$), since each pair of values must appear exactly once. Hence, each orthogonality constraint is of arity 4.

Definition 5.2 *A* label *is a variable-value pair $< x_i, v_i >$ representing the assignment of value v_i to variable x_i. A label is meaningful if and only if $v_i \in D_i$. We normally say that variable x_i is* instantiated *to value v_i. A* compound label *is a set $\{< x_1, v_1 >, < x_2, v_2 >, \ldots, < x_k, v_k >\}$ representing the simultaneous assignment of the tuple $\{v_1, v_2, \ldots, v_k\}$ to the variables $\{x_1, x_2, \ldots, x_k\}$. Under this perspective, a constraint can alternatively be viewed as a set of compound labels. A compound label $\{< x_{j_1}, v_{j_1} >, \ldots, < x_{j_m}, v_{j_m} >\}$ satisfies a constraint $C(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$ if and only if $(v_{j_1}, v_{j_2}, \ldots, v_{j_m}) \in C(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$.*

Definition 5.3 *A solution to CSP(X,D,C) with variables $x_1, x_2, \ldots, x_n$ is a compound label $\{< x_1, v_1 >, < x_2, v_2 >, \ldots, < x_n, v_n >\}$, where $v_i \in D_i$, for all $i \in \{1, ..., n\}$, and $(v_{j_1}, v_{j_2}, \ldots, v_{j_m}) \in C(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$ for all $C(x_{j_1}, x_{j_2}, \ldots, x_{j_m}) \in C$.*

When solving a CSP, we have to conduct search for one of the following:

- A single solution illustrating that the problem is feasible
- A subset, or possibly all, of the solutions to identify the optimal one.
- The entire solution space, to prove that the problem is infeasible.

5.1.3 The constraint graph

Definition 5.4 *A graph is a tuple $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges.*

Definition 5.5 *A hypergraph is a tuple $H = (V, E)$, where V is the set of nodes and E is the set of hyperedges. A hyperedge is a subset of V of arbitrary cardinality.*

Each CSP can be associated with a hypergraph, having one node for each variable and one hyperedge for each constraint. A hyperedge includes the nodes corresponding to variables involved in the constraint. Hence, the hypergraph does not provide any information about the domains of the variables or the set of tuples allowed by each constraint. If the arity of each constraint is 2 (binary CSP), the hypergraph reduces to a graph.

Example 5.2 *First, consider the constraint graph for the problem of finding a single Latin square. Actually, the problem can be naturally formulated as a graph-colouring problem, if the domain members are viewed as colours ([60]). Let $G_1 = (V_1, E_1)$ be this graph. Each node v corresponds to a variable X_{ij}, whereas each edge denotes that its incident nodes cannot have the same colour. Therefore, $V_1 = \{X_{ij}, i, j = 0, \ldots, n-1\}$,*

$$E_1 = \{(X_{i_0 j_0}, X_{i_1 j_1}) : i_0 = i_1 \text{ or } j_0 = j_1,\ i_0, i_1,\ j_0, j_1 \in \{0, ..., n-1\}\}$$

where $|V_1| = n^2$ and $|E_1| = n^2(n-1)$. The constraint graph $G_2 = (V_2, E_2)$, associated with the fact that Y must also be Latin is a replicate of G_1.

Additional edges are necessary to formulate the orthogonality constraints. Let $G_3 = (V_3, E_3)$ be the constraint hypergraph, with $V_3 = V_1 \cup V_2$, $E_3 = E_1 \cup E_2 \cup E_0$, where E_0 consists of the hyperedges corresponding to orthogonality constraints. Each such hyperedge involves four nodes and it is straightforward to see that:

$$E_0 = \{(X_{i_0 j_0}, Y_{i_0 j_0}, X_{i_1 j_1}, Y_{i_1 j_1}) : i_0 \neq i_1 \text{ or } j_0 \neq j_1,\ i_0, i_1,\ j_0, j_1 \in \{0, ..., n-1\}\}$$

where $|E_0| = \frac{n^2(n^2-1)}{2}$. Overall, $|V_3| = 2n^2$, $|E_3| = 2n^2(n-1)+ \frac{n^2(n^2-1)}{2}$.

5.1.4 Arc-consistency and its generalisations

Identifying a solution to a CSP involves a certain search procedure, meaning that different values have to be assigned to different variables in a systematic way, until a solution is found. If at some point the search is no longer possible, we have to backtrack to a previous step and examine the next value available.

In order to accelerate the process, it would be useful to identify values in the variable domains, which could never be part of a solution. Removing these inconsistent values eliminates redundant search. This idea is the motivation for a general approach to solve a CSP, called *constraint propagation.* The notion of *arc-consistency* (AC), defined below, is the simplest form of constraint propagation.

Definition 5.6 *Assume a binary CSP and the associated constraint graph. An arc (v_j, v_k) is consistent, if, for every value a in D_j there is some value b in D_k, such that $(a, b) \in C(x_j, x_k)$. The value b can be viewed as a* support *for value a and vice versa.*

Note that this definition implies that the notion of arc-consistency is directional, meaning that arc (v_j, v_k) being consistent does not necessarily imply that arc (v_k, v_j) is consistent too. In most problems, however, the relationship is bidirectional, i.e. (v_j, v_k) is consistent if and only if (v_k, v_j) is consistent. In these cases, one can refer to *edge-consistency.* It is easy to see that OLS falls within this category.

Example 5.3 *Consider the case from the constraint graph of the single Latin square problem, as illustrated in Fig.5.1.*

It is easy to see that both edges (x_{13}, x_{15}) and (x_{14}, x_{15}) are not consistent, since value 2 in both D_{13} and D_{14} does not have any supportive value in D_{15}. Hence, value $_2$ must be deleted from both D_{13} and D_{14}.

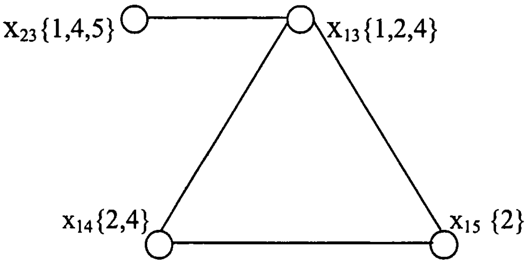


Figure 5.1: An example of arc-consistency

In order to achieve edge-consistency in a constraint graph, all arcs have to be examined. For each $(v_j, v_k) \in E$, the values of X_j, which have no support in D_k, are deleted from D_j. Deleting such a value can have further implications for the domains of all variables V_t, such that $(V_t, V_j) \in E$. Therefore, the above procedure has to be repeated until no more domain reduction is possible.

Example 5.3 (cont.) *In the above example, the status of the domains after deleting value* 2 *from* D_{13} *and* D_{14} *is exhibited in Fig.5.2. It is now obvious that edge* (x_{13}, x_{14}) *is no longer consistent*

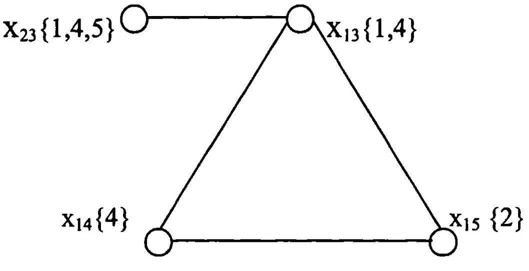


Figure 5.2: Second step in achieving arc-consistency

and 4 *has to be removed from* D_{13}*. Once this takes place,* $D_{13} = \{1\}$*. Observe that the value* 1 *in* D_{23} *has no support in* D_{13}*, hence it must also be removed. All the edges are then consistent, and the condition is depicted in Fig.5.3.*

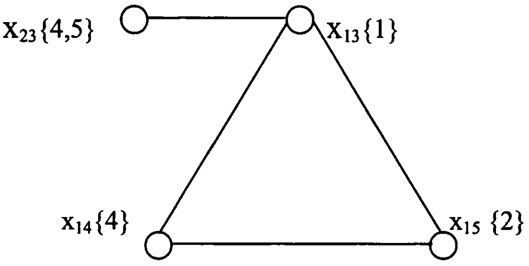


Figure 5.3: Arc consistent subgraph

Although in the above example the edges can become consistent by inspection, this is clearly not possible in larger problems. Certain algorithms have been proposed for making a constraint graph arc-consistent. Algorithm 5.1 presents an outline of the *AC-3* procedure for arc-consistency ([83, Section 4.2.3]) in pseudo-code.

Algorithm 5.1 (Algorithm AC-3)

```
$Q = \{(v_j, v_k) \in E, i \neq j\}$
repeat
{
    CHANGE ← false;
    for each $(v_j, v_k) \in Q$ do
    {
        for each $a \in D_j$ do                    // $D_j$ is the domain of variable $j$
        {
            DELETE ← false;
            if, for each $b \in D_k$, $(a,b) \notin C(x_j, x_k)$      // No support value is left for a
            {
                Delete a from $D_j$;
                DELETE ← true;
            }
            CHANGE ← CHANGE ∨ DELETE;
        }
    }
until (CHANGE =false)                    // Until no more values can be deleted
return;
```

Note that Algorithm 5.1 assumes directionality. The authors of [14] present a variant of *AC-3*, which takes bidirectionality into account, hence reducing the computation needed for maintaining arc-consistency. More efficient algorithms, namely *AC-4* & *AC-5,* are presented in [67] and [49], respectively. These schemes can maintain arc-consistency in $O(ed^2)$ steps in the worst case, where d is an upper bound on the size of each domain and e is the number of constraints.

The question, which naturally arises, is whether, after making the constraint graph arc-consistent, no search is required to advance to a solution. If this is indeed the case, the constraint graph is said to have become *backtrack-free.* This could occur only if:

- Every variable is left with a single value in its domain, hence indicating a unique solution.
- The domain of a variable is left empty, therefore the CSP is infeasible.
- All the values left in the domains can be part of a solution, therefore any combination of these remaining values can be used to construct a solution. Such a case is illustrated in Fig.5.3.

Unfortunately, the above conditions rarely occur and only in special problem instances. A more formal argument is provided by the notion of *k-consistency,* which generalises that of arc-consistency

Definition 5.7 *A graph is* k-consistent, *if, whenever a partial instantiation of any $k-1$ variables violates none of the constraints involving those variables, there exists a value for any other kth variable, which violates none of the constraints involving all k variables. A graph is* strongly k-consistent *if it is j-consistent for all $j \leq k$.*

Hence, arc-consistency is equivalent to 2-consistency. *Path-consistency* is an extension of arc-consistency, equivalent to 3-consistency. In path-consistency, any two binary constraints with one variable in common are also examined. For example, assume the existence of constraints $C(x_j, x_k)$

and $C(x_k, x_l)$. The domains of x_j & x_l are indirectly interrelated, and this implicit information can be used to exclude values from both D_j & D_l. Although only binary constraints are included in the original problem, path-consistency can induce implicit constraints of arity 3, in the form of triples of (x_j, x_k, x_l) that can be part of a solution.

Clearly, no search is necessary if a graph can be made strongly $|V|$-consistent, since no constraint can involve more than $|V|$ variables. An optimal algorithm for achieving that is proposed in [28]. However, achieving strong $|V|$-consistency can be computationally more expensive than the most naive search procedure. There are specific problem classes, for which the constraint graph can become backtrack-free, if it is made k-consistent for a certain $k < |V|$. As it is usually the case, the methods achieving this are problem-specific and therefore difficult to generalise ([57]).

5.2 Methods for solving a CSP

This section examines a number of representative algorithms for solving CSPs. The algorithms are presented in ascending order with respect to their level of sophistication. An excellent survey of CSP algorithms can be found in [57]. The authors of [56] provide a theoretical framework for assessing the performance of most of the schemes presented here.

5.2.1 Generate & Test

Generate & Test (GT) is the simplest method for solving a CSP. All possible assignments of values to variables are generated in a systematic way. After a complete assignment (compound label) is generated, it is tested against all problem constraints. If none of those is violated, the assignment is a feasible solution and the algorithm terminates. If a violation occurs, the next assignment is generated. The problem is declared infeasible, if all possible assignments fail to give a solution.

If the problem is defined on m variables, each with a domain of size p, m^p assignments have to be generated if (i) no solution exists or (ii) all solutions are required or (iii) the best solution is required. Assume a binary constraint $C(x_j, x_k)$ not allowing the compound label $(< x_j, a >, < x_k, b >)$. All n^{m-2} assignments containing this label will be generated, although this is obviously fruitless.

Example 5.4 *Suppose that we wish to solve the Latin square problem with GT. There can be no Latin square of order n having two 0's in the first column, therefore no assignment of the form* $(< X_{00}, 0 >, < X_{10}, 0 >, \ldots)$ *can be a solution. There exist, however,* n^{n^2-2} *such assignments, which will be generated and tested. The amount of redundant search becomes more significant, if all such cases are taken into account.*

5.2.2 Chronological Backtracking

A normal extension of the above technique is to proceed incrementally with variable instantiation. This means that, at each step, only one variable is assigned a value and all the constraints are checked with respect to the variables instantiated so far. If a violation occurs, the most recently instantiated variable, which has still values not considered, is re-instantiated to its next available value. Thus, partial instantiation is sufficient to prune larger subsets of the solution space. This constitutes the *Chronological Backtracking* (BT) paradigm.

The search performed by BT can be represented by a search tree. Each level in this tree corresponds to a variable and nodes at the same level correspond to different values assigned to the particular variable. At the top node of the tree, no variable has been instantiated.

Example 5.5 *Part of the search tree for the problem of finding a Latin square of order* 3 *is shown in Fig.5.4. It is assumed that the variables are instantiated in the order* X_{00}, X_{01}, X_{02} *and so on, i.e. we fix rows of the Latin square.*

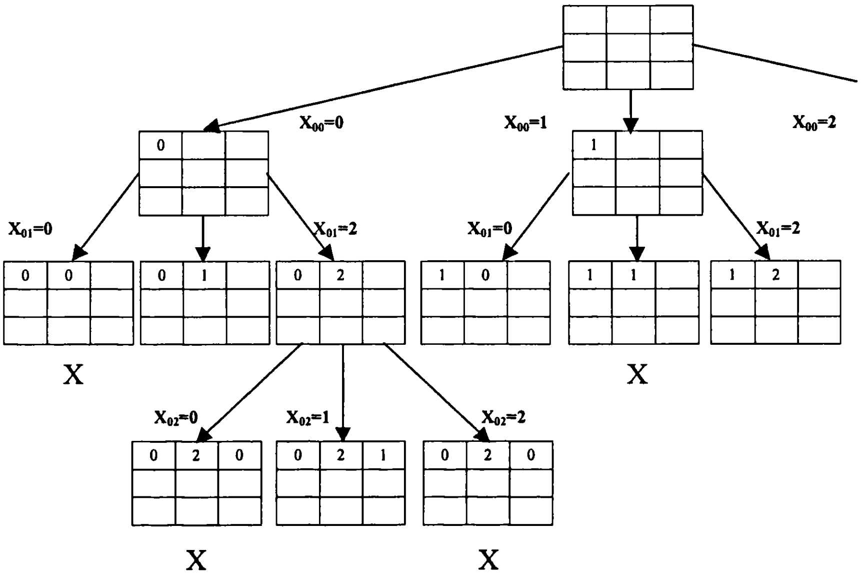


Figure 5.4: An example of a BT search tree

BT examines the nodes of the tree according to a *depth-first* strategy, as described by the steps of the following algorithm.

Algorithm 5.2 (Chronological Backtracking)

Step I *Select a variable, which has not yet been assigned a value, i.e. define a new level in the tree. If there are none stop; a solution has been found.*

Step II *Select a value for this variable (a node in that level), not yet examined. If such a value exists, go to Step 3. If not, backtrack to the most recently instantiated variable, which still has values not examined (backtrack to the deepest previous level, which still has nodes not examined). If no such variable exists, stop; the problem is infeasible.*

Step III *Test whether the new partial solution satisfies all the constraints involving the variables instantiated up to this point. If so, go to Step 1; otherwise, go to Step 2.*

It is not difficult to see that algorithm BT behaves more efficiently than GT. For the example of Fig.5.4, all assignments having $X_{00} = 0$ & $X_{01} = 0$ are excluded from further consideration already at level 2. However, this holds only because variable X_{01} is instantiated right after X_{00}. Let us consider a different case.

Variables X_{00} and X_{30} must also possess different values, hence the label $< X_{00}, 0 >$ implies that the label $< X_{30}, 0 >$ will lead to a violation. Assume a point in the search procedure, where value 0 is the only one available for variable X_{30}. Notice that 6 variables have been already instantiated, i.e. variables X_{00}, X_{01}, X_{02}, X_{10}, X_{11}, X_{12}. Given that the label $< X_{30}, 0 >$ is not allowed, the algorithm will have to backtrack to the most recently instantiated variable having still available values. However, only backtracking all the way to X_{00} could allow for a new value to become available for X_{30}. Until this takes place, all available values for the 5 intermediate variables have to be examined. Hence, a significant amount of redundant search is performed. This phenomenon, arising from having a CSP that is not arc-consistent, is called *thrashing.*

5.2.3 Forward Checking

BT examines each label with respect to all previously created labels (i.e. variable instantiations) and rejects the current label, if it is inconsistent with a previous one. In this sense, it performs consistency checks only between instantiated variables. However, possible inconsistencies can be detected, and thus avoided, much earlier. Let $(a, b) \in C(x_j, x_k)$. Setting $x_j = a$ should automatically exclude the value b from D_j. This simple idea is adopted by the *Forward Checking* (FC) scheme, which avoids useless assignments by *looking ahead* at the effect of any current assignment on uninstantiated (future) variables. FC creates a search tree, exactly as BT. The difference is that, instead of examining the constraints regarding only instantiated variables, it also uses the constraints to delete values from the domains of the uninstantiated variables. The main steps of the FC scheme are illustrated in Algorithm 5.3.

Algorithm 5.3 (Forward Checking)

Step I *Select a variable, which has not yet been assigned a value. If there are none stop; a solution has been found.*

Step II *Select a value still in the domain of this variable, which has not yet been examined. If such a value exists, go to Step 3. If not, backtrack to the previously instantiated variable, which still has values not examined. If no such variable exists, stop and declare the problem infeasible.*

Step III *Delete any inconsistent values from the domains of the uninstantiated variables. If a domain is annihilated during this process, go to Step 2. If not, go to Step 1.*

Essentially, Step III performs domain reduction by examining all constraints involving the currently selected variable.

Example 5.5 (cont.) *In this example, the FC algorithm would detect that, once $X_{00} = 0$, value 0 should be excluded from the domains of all variables having one index in common with X_{00}. Assuming that the search proceeds by fixing rows, we present a part of the search tree in Fig.5.5. In each node, the variable domains are also illustrated in brackets.*

At each node, it is no longer important to check the current variable x_k against all previous variables, since inconsistent values have already been removed from D_k. Extra work is necessary only for filtering the domains of the remaining variables. By comparing Figures 5.4 & 5.5, one can already see that FC creates a smaller search tree than BT. It has been formally proved ([56]) that, for any CSP and for identical variable orderings, FC creates at most as many nodes as BT.

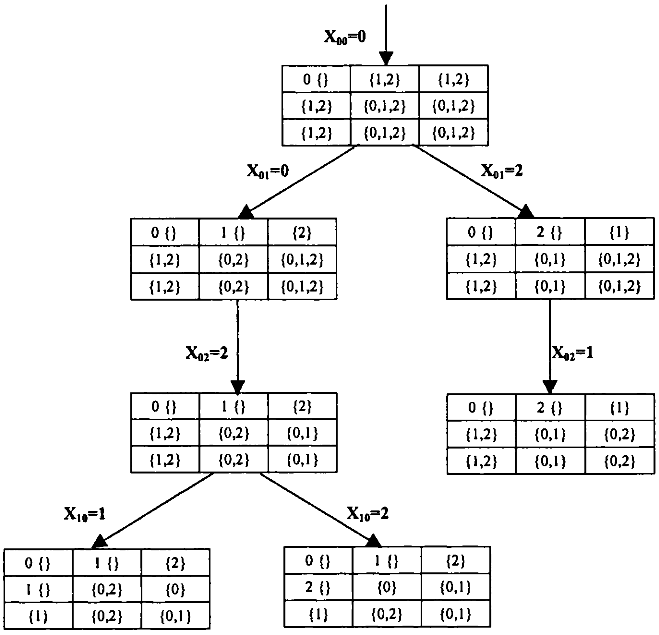


Figure 5.5: The search tree for FC

Let us provide a rather artificial example, illustrating why BT may perform significantly more work that FC. For a Latin square of order $n > 2$, consider the variables $X_{00}, X_{01}, X_{0(n-1)}$, where $D_{00} = D_{01} = D_{0(n-1)} = \{0, 1\}$. Since all these variables must be pairwise different, the problem is obviously infeasible. BT will assign different values to the first two variables, say $< X_{00}, 0 >, < X_{01}, 1 >$, and will keep on assigning values to all intermediate variables, until it reaches $X_{0(n-1)}$ and discovers that no value can be assigned to it. It will then gradually backtrack and consider all possible combinations of the intermediate variables, before tracing the source of the problem and declare the problem infeasible. In contrast, FC will detect infeasibility right after instantiating X_{00} & X_{01} because $D_{0(n-1)}$ becomes empty. Hence, FC will never instantiate any intermediate variables.

5.2.4 Forward schemes

The previous sections indicate that a spectrum of different methods can be applied to solve CSPs. The issues to be addressed by every method are the following.

- The order, according to which variables are instantiated. We have implicitly assumed that this order is given at the beginning of the algorithm.
- The level of constraint propagation to be performed after fixing a variable.
- The variable to which to backtrack if no more values are available for the current variable or if a domain is annihilated.

Schemes, which mainly focus on addressing the second issue, are called *forward* or *look-ahead*, whereas algorithms based on resolving the third one are called *backward*.

The difference among forward schemes lies in the level of constraint propagation performed at each node. The main trade-off is between the aggressiveness of constraint propagation and the size of the search tree. An exhaustive algorithm for k-consistency can significantly reduce the tree size, posing more computational effort at each node. The relevant literature depicts studies, where such an aggressive strategy performed less efficiently than even BT (see [56, 57] for more details). Another variant might be that the level of constraint propagation increases or decreases with respect to the depth of the node. Applying more extensive constraint propagation in the higher tree levels can drastically reduce the size of variable domains, leaving little further work even for a simple BT scheme. Notice that even BT incorporates an elementary form degree of constraint propagation, since any assignment to a variable of a value inconsistent to the already instantiated variables causes immediate failure.

FC is a forward scheme, which performs consistency checks only between instantiated and uninstantiated variables. The *Full Look-Ahead* (FLA) algorithm performs further domain reduction than FC, by looking for inconsistencies even between the domains of uninstantiated variables. Essentially, FLA ensures arc-consistency at each node of the tree. For example, consider the condition after the assignments $< X_{00}, 0 >, < X_{01}, 1 >, < X_{02}, 2 >, < X_{10}, 1 >$ in Figure 5.5. FLA would go one step further than FC and deduce that $D_{20} = \{1\}, D_{22} = \{0\}, D_{21} = \{2\}$. At this point, assigning to each variable the unique value in its domain attains a solution.

5.2.5 Backward schemes

Backward schemes are also named Intelligent Backtracking or Gather-Information-While-Searching strategies [83]. They differ from all schemes discussed so far in the fact that they do not necessarily backtrack to the most recently instantiated variable.

Suppose that variables $x_1, x_2, \ldots, x_{k-1}$ are instantiated, in this order, and that the instantiation of x_k fails. BT will backtrack to x_{k-1} and examine a different value from D_{k-1}. However, this might not affect the failure of assigning a value to x_k. The simplest reason might be that there is no constraint involving both x_{k-1} and x_k. Therefore, changing the value of x_{k-1} will not affect the failure in the next step.

Instead, the algorithm could trace the most recently instantiated variable, which caused the failure of at least one of the possible assignments to x_k. This idea is implemented by the *Back-Jumping* (BJ) algorithm, which "jumps" directly to that variable. Let this variable be x_j, $1 \leq j \leq k-2$. BJ cancels all intermediate assignments to variables $x_{j+1}, \ldots, x_{k-1}$. The natural question is what should happen, if no further value is available for x_j either. The obvious resolution is to repeat the process for x_j and move to variable, say, x_i, $1 \leq i \leq j-1$. The problem is that our initial aim was to identify the reason for the failure of x_k. If another variable x_m, $i < m < j$ is also responsible for the failure of x_k, altering its value could make the instantiation of x_k possible. If this is the case, the algorithm has just omitted a feasible solution. In order to overcome this effect, the BJ algorithm performs a jump initially and backtracks chronologically thereafter.

An improved version of BJ is the *Conflict-Directed Back-Jumping* (CBJ). In this scheme, each variable has an associated constraint set, which includes all the preceding variables that caused failure during this variable's instantiation. In this sense, the algorithm records the reasons of failure and learns from the past. When no more values remain for the current variable x_k, the algorithm backtracks to the deepest variable of its constraint set. Let this variable be x_j. If instantiation of x_j to another value is also impossible, the constraint sets of x_k & x_j are merged. The algorithm

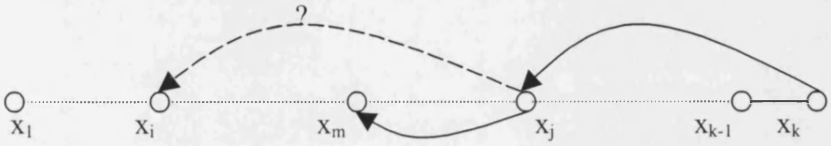


Figure 5.6: The BJ scheme

backtracks to the most recent variable in this combined set, i.e. the most recent variable, which has caused the failure of x_j or x_k. Therefore, no information is lost if two successive jumps take place. In [56], it is proved that CBJ creates at most as many nodes in the search tree as BT and BJ. The relationship between the number of nodes created by CBJ and FC remains unknown.

5.2.6 Hybrid methods

Backward and forward schemes can be integrated. The most promising hybrid is the FC-CBJ algorithm, which combines the advantages of both FC and CBJ. A theoretical evaluation of algorithms for CSP proves that FC-CBJ is at least as competitive as any other known algorithm in terms of the size of the search tree.

FC-CBJ can be described a standard FC scheme, where, in addition, each variable has an associated constraint set. Each time that the instantiation of a variable x_j reduces the domain of a uninstantiated variable x_k, $j < k$, x_j is added to the constraint set of x_k. If the domain of x_k is annihilated, the conflict set of x_k is added to that of x_j and the algorithm backtracks to the most recent variable in the new set, say $x_i, i < j$. All intermediate assignments are also cancelled. The extra task is to restore the domains of the uninstantiated variables in the state before the instantiation of variables $x_{i+1}, \dots x_j$.

The basic concern about the performance of hybrid schemes is whether the combined approaches are complementary or overlapping. CBJ offers a significant improvement over BT, but, once added to a FC or FLA scheme, it possibly increases the solution time, although the tree size is reduced (see [57, 83]). Essentially, it is difficult for CBJ improve significantly over an FC algorithm. Preliminary experimentation has shown the same result concerning the OLS problem. To the best of our knowledge (see also [83, 84]), the relevant literature includes no study where backward schemes outperformed FC or FLA algorithms.

5.2.7 Variable and value ordering

The previous sections focused mainly on the procedure to be followed at each node of the search tree, the goal being to overcome the weaknesses of backtracking and to reduce the tree size by eliminating redundant search. The ordering of the variables has been assumed to be known in advance, although this is obviously not the case. Successfully selecting the next variable to be instantiated (also called *branching* variable) and the node to be processed first can affect dramatically the overall performance. Ideally, if a solution to the original CSP exists, instantiating each variable to its correct value can lead to the solution in linear time with respect to the number of variables, without involving any backtracking.

Several schemes for variable selection have been studied, where the order of variables instantiation is determined either *statically*, i.e. only once in the root node, or *dynamically* at every node. A simple approach suggests to branch on the variable with the smaller domain cardinality. This dynamic method creates the minimum number of new nodes at each step. Another method is to branch on the variable participating in the largest number of constraints. In this case, the order of variables can be determined initially. The intuition behind this method is that, since the largest possible number of constraints will be used for propagation at each step, the domain reduction will be rigorous. Hence, branches leading to infeasibility will be pruned earlier in the search. An extension of this idea is to branch on variables in the way that maximises the number of domain members removed from the uninstantiated variables. This approach is expected to reduce the tree size and accelerate the search ([83, Chapter 6]).

A more sophisticated scheme makes use of the *stable set* of variables. A stable set in a graph is defined as a set of nodes with no edges between any two of them. Similarly, a stable set in a (binary) CSP is a set of variables having no constraints involving any two of them. If the maximum stable set of a given CSP was known, the variables of the set could be the last to be instantiated, since we know that constraint propagation will not produce any reduction for all the other variables in the set. Unfortunately, determining the maximum stable set is an $\mathcal{NP}$-hard problem. Heuristics can be possibly applied to find a sub-optimal solution, as long as this involves a sufficiently large fraction of the variable set.

The ordering of values for a certain variable remains an unexplored issue. Estimators have been used to calculate the virtue of each domain member. Common criteria are to prefer the value, which maximises the options available for future assignments, or to prefer the value, which is expected to lead to the easiest to solve CSP. The setback is again that all the developed heuristics remain highly problem specific.

5.3 CP algorithms for the OLS problem

5.3.1 Problem formulation

Let I, J be the n-sets indexing the rows and columns of squares X and Y. Define variables X_{ij}, Y_{ij} as in Example 5.1. Recall from Section 5.1.3 the constraint graph for the Latin square problem $G_1 = (V_1, E_1)$, where

$$\begin{aligned} V_1 &= \{X_{ij}, i = 0, \ldots, n-1, j = 0, \ldots, n-1\} \\ E_1 &= \{(X_{i_0 j_0}, X_{i_1 j_1}) : i_0 = i_1 \text{ or } j_0 = j_1,\ i_0, i_1,\ j_0, j_1 \in \{0, ..., n-1\}\} \end{aligned}$$

It is easy to verify that the node subsets $\{X_{0j}, \ldots, X_{(n-1)j}\}$, for $j = 0, \ldots, n-1$ and $\{X_{i0}, \ldots, X_{i(n-1)}\}$, for $i = 0, \ldots, n-1$ are (maximal) cliques, since any two nodes in each subset are connected and no other node connected to all existing ones can be identified. Each clique can be naturally represented in CP by the *all_different* operator ([53, 77]). The constraint *all_different*(S), where S is a set of variables, signifies that all variables in S must be assigned pairwise different values. It is easy to see that the constraint *all_different*(S) is equivalent to a set of $\frac{|S|\cdot(|S|-1)}{2}$ binary constraints of the form $x_i \neq x_j$, $i, j \in S$, $i \neq j$. Given that we have to impose an *all_different* constraint for each row and column of Latin square X, the CP formulation of the Latin square problem is the following.

$$\begin{aligned} &all\_different\{X_{ij} : i \in I\}, \text{ for all } j \in J \\ &all\_different\{X_{ij} : j \in J\}, \text{ for all } i \in I \\ &D_{X_{ij}} = D_X = \{0,\dots,n-1\} \end{aligned} \tag{5.12}$$

Let us consider the OLS problem, where X and Y denote the two squares. Formulation (5.12) must be repeated for square Y and the associated constraint graph $G_2 = (V_2, E_2)$ is a replicate of G_1. The orthogonality constraints correspond to hyperarcs, each one involving four nodes. Let $G_3 = (V_3, E_3)$ be the constraint graph. As discussed in Section 5.1.3,

$$\begin{aligned} V_3 &= V_1 \cup V_2 \\ E_3 &= E_1 \cup E_2 \cup E_0 \\ E_0 &= \{(X_{i_0j_0}, Y_{i_0j_0}, X_{i_1j_1}, Y_{i_1j_1}) : i_0 \neq i_1 \text{ or } j_0 \neq j_1,\ i_0, i_1,\ j_0, j_1 \in \{0,...,n-1\}\} \end{aligned}$$

where $|E_0| = \frac{n^2(n^2-1)}{2}$. It follows that $|V_3| = 2n^2$, $|E_3| = 2n^2(n-1) + \frac{n^2(n^2-1)}{2}$.

In order to formally express the orthogonality constraints, define the auxilliary variables $Z_{ij} = X_{ij} + n \cdot Y_{ij}$, for $i, j = 0, 1, \dots, n-1$. There are n^2 possible values for Z_{ij}, i.e. $D_{Z_{ij}} = D_Z = \{0, \dots, n^2-1\}$, which have a $1-1$ correspondence with all n^2 ordered pairs (i, j), for $i, j = 0, 1, \dots, n-1$. Each ordered pair appears exactly once if and only if all Z_{ij}s are assigned pairwise different values. Therefore, the OLS problem can be formulated as follows.

$$\begin{aligned} &all\_different\{X_{ij} : i \in I\}, \text{ for all } j \in J \\ &all\_different\{X_{ij} : j \in J\}, \text{ for all } i \in I \\ &all\_different\{Y_{ij} : i \in I\}, \text{ for all } j \in J \\ &all\_different\{Y_{ij} : j \in J\}, \text{ for all } i \in I \\ &all\_different\{Z_{ij} : i \in I, j \in J\} \\ &Z_{ij} = X_{ij} + n \cdot Y_{ij}, \text{ for all } i \in I, j \in J \\ &D_X = D_Y = \{0,\dots,n-1\}, D_Z = \{0,\dots,n^2-1\} \end{aligned} \tag{5.13}$$

This formulation involves $2n^2$ variables with domains of cardinality n and n^2 variables with domains of size n^2. It also includes $4n + 1$ *all_different* constraints and n^2 equalities. Overall, formulation (5.13) requires $3n^2$ variables and $n^2 + 4n + 1$ constraints.

Notice that the entities of rows, columns and values in the two squares can be viewed interchangeably, exactly as in the IP model. Since any two of these entities are indexed by i & j in formulation (5.13), $\binom{4}{2} = 6$ possible representations are possible. To illustrate the symmetry implied by such a perspective, 6 equivalent problems, having the same constraint graph, can be stated:

I. (i indicates the row and j the column): pair (X_{ij}, Y_{ij}) appears in cell (i, j);

II. (i indicates the row and j the value of the first square): pair (j, Y_{ij}) appears in cell (i, X_{ij}). Alternatively stated, X_{ij} is the column, where values j and Y_{ij} occur in row i of the first and the second square, respectively;

III. (i indicates the row and j the value of the second square): pair (X_{ij}, j) appears in cell (i, Y_{ij});

IV. (i indicates the column and j the value of the first square): pair (j, Y_{ij}) appears in cell (X_{ij}, i);

V. (i indicates the column and j the value of the second square): pair (X_{ij}, j) appears in cell (Y_{ij}, i);

VI. (i indicates the value of the first square and j the value of the second square): pair (i, j) occurs in cell (X_{ij}, Y_{ij}).

The remainder of this section is organised as follows: Section 5.3.2 discusses possible branching rules, while Section 5.3.3 presents the actual CP algorithms and proposes methods for Constraint Propagation. The terminology follows that of Section 4.4.

5.3.2 Branching strategies

Before the search starts, the preliminary variable fixing implied by Table 4.2 is implemented. This amounts to fixing the first rows of squares X, Y and the first column of X in natural order, along with further filtering the domains in the first column of Y. Given this starting point, this section discusses the order, according to which the remaining variables/cells will be instantiated. The first cells to be fixed are the remaining cells of the first column of square Y. Concerning the remaining $(n-1) \times (n-1)$ subsquares, there are two systematic ways for branching, namely fixing rows (columns) in both squares or fixing transversals (see Section 4.4.2).

Moreover, a critical question is whether the two squares will be fixed concurrently or X will be completely fixed before starting to fix Y. One approach is to first assign values to square X and then to square Y. This branching rule implies that X will be instantiated to different Latin squares of order n up to the first square, which does have an orthogonal mate. Square X (and Y, afterwards) can be constructed by tentatively fixing either rows or sets of cells in pairwise different row and column. In the latter case, the cells must be assigned identical values, hence this second method fixes values rather than cells. In either case, the Latin squares will be produced according to a certain lexicographic rule. For example, in the case of fixing rows, all squares X having value 0 in cell $(1,1)$ will be produced before the squares having 0 in cell $(1,2)$. Therefore, the success of a method fixing square X completely before examining square Y depends highly on whether a Latin square having an orthogonal mate appears early in this order. If not, numerous Latin squares of order n will have to be pointlessly generated. Apparently, the lexicographic order is not important for $n = 6$. On the other hand, this approach sounds simpler and has the advantage of over-constraining the variables Y. In more detail, variables X must satisfy only the constraints (5.12), whereas variables Y are additionally constrained by the orthogonality constraints, i.e. they have to satisfy all constraints (5.13). Hence, for given X, it is expected that checking the existence of an orthogonal Y should be simpler. Notice that this approach relates to model $(IP2)$ of Section 4.1.

If we wish to assign values to both squares at the same time, we have to first select a cell (i, j), then instantiate variable X_{ij} to the smallest available value in its domain and directly proceed with the instantiation of variable Y_{ij}. Exactly as in Section 4.4.2, we examine the strategies of fixing either rows or transversals of Y. Although the analysis of Section 4.4.2 suggests that fixing transversals is a superior option, we will still experiment with both branching schemes. The four branching rules, applied in the CP algorithms of the next section, are the following.

X_R static variable ordering, where the variables of square X are instantiated first by fixing rows. Variables of square Y instantiated afterwards, also by fixing rows.

X_T: dynamic variable ordering, where the variables are instantiated by fixing sets of cells in pairwise rows and columns. Hence, all 0s are first fixed in proper cells of X, followed by all 1s and so on. Once square X is completed, the same rule is applied to square Y.

XY_R static variable ordering, fixing rows in both squares. Hence, cell $(1,1)$ is the first to be assigned a pair, followed by cell $(1,2)$ and so on.

XY_T: dynamic variable ordering, fixing transversals of Y along by fixing all corresponding cells of X to identical values. Again, value 0 is the first to be fixed in n cells of X, each in a different row and column, followed by value 1 and so on.

Note that the last two strategies start by fixing the first column of square Y. The last note is that all CP algorithms follow a *depth-first* policy for node selection, again because the search is about a single feasible solution.

5.3.3 Algorithms & Constraint Propagation

GT algorithms are not considered because of their apparent inefficiency. The BT algorithm offers simplicity and fast enumeration, since it does not apply any form of constraint propagation. For this reason, it is included in our experimentation.

The BT algorithm, combined with branching rule X_R, will select the next cell in the current row i_0, instantiate variable $X_{i_0j_0}$ to value k_0. It will then check whether value k_0 has already been assigned to any variable in row i_0 and columns $j = 0, ..., j_0 - 1$ or any variable in column j_0 and row $i = 0, ..., i_0 - 1$. If this is true, the algorithm backtracks and selects the next available value for variable $X_{i_0(j_0-1)}$, if $j_0 \geq 2$ or for variable $X_{(i_0-1)(n-1)}$, if $j_0 = 1$. When variable $Y_{i_0j_0}$ is instantiated to value l_0, the algorithm again looks for the possible appearance of value l_0 at cells of either the same column and all previous rows or the same row and all previous columns. Additionally, it considers the value of variable $X_{i_0j_0}$. If this value is k_0, the algorithm will have to check all cells of square X containing this value and look at whether value l_0 appears in any of the corresponding cells of square Y. This will ensure that pair (k_0, l_0) will appear exactly once. In order to accelerate this process, we introduce the auxilliary binary variables $Pair_{kl}$, which essentially model variables Z_{ij} of (5.13). When $Y_{i_0j_0}$ is set to l_0, only variable $Pair_{k_0l_0}$ needs to be checked. If $Pair_{k_0l_0} = true$, a violation of orthogonality has occurred; otherwise, $Pair_{k_0l_0}$ is set to value *true* to acknowledge the existence of pair (k_0, l_0) to any future assignments, i.e. descendant nodes. It is not difficult to see that all consistency checks can be implemented in $O(n)$ steps. These checks are performed regardless of the branching rule.

The BT algorithm leaves the variable domains intact throughout the whole solution process. For this reason, it performs redundant search, since some values are already inconsistent because of previous assignments. If, for example, $X_{i_0j_0} = k_0$, value k_0 is already forbidden for all other $X_{i_1j_1}$, such that $i_0 = i_1$ or $j_0 = j_1$. This redundancy can be avoided, if we perform domain reduction at each step. In this way, each time a variable is instantiated to a value still existing in its domain, no constraints have to be checked. This direction leads, apparently, to a Forward Checking algorithm.

Hence, algorithm FC fixes a variable at the smallest value still existing in its domain. Its next step is to look at the uninstantiated variables and delete any inconsistent values from their domains. If, during this process, a domain is annihilated, the current node is declared infeasible. If the domain is left with a single value, the corresponding variable is directly instantiated and the process of domain reduction is invoked again. Hence, variables are generally instantiated either because they are currently selected or as a consequence of domain filtering.

More formally, if variable $X_{i_0j_0}$ is instantiated to value $k_0 \in D_{X_{i_0j_0}}$, k_0 is removed from the domains $\{D_{X_{i_1j_1}} : i_1 = i_0$ or $j_1 = j_0$, $X_{i_1j_1}$ is uninstantiated$\}$. If $D_{X_{i_1j_1}} = \emptyset$, the node is pruned and the algorithm backtracks to its immediate predecessor. If $D_{X_{i_1j_1}} = \{k_1\}$, $X_{i_1j_1}$ is instantiated to value k_1 and the previous procedure is performed recursively. Notice that this procedure is analogous to the preprocessing process for the B&C algorithm (Section 4.4). Identical steps are followed for square Y. In order to reduce domains based on orthogonality constraints, setting $X_{i_0j_0} = k_0$ implies that value k_0 has to be erased from all the domains $\{D_{X_{i_1j_1}}$: $Y_{i_1j_1}$ is instantiated, $Y_{i_1j_1} = Y_{i_0j_0}\}$, the same being applicable to square Y. In order to check which variables $X_{i_1j_1}$ have values equal to $X_{i_0j_0}$, we retain points $PT_{i_0k_0}$ for each row i_0 and value k_0, which indicate the cell of row i_0 where value k_0 (possibly) occurs. The domain reduction procedure is performed in $O(n)$ steps. Observe, however, that under the branching scheme *XY_T* only cells belonging to the same transversal need to be examined for square X, since only these cells can contain the same value as $X_{i_0j_0}$. These variables have been fixed just before $X_{i_0j_0}$, thus any violation of orthogonality can be resolved by backtracking for up to $(n-1)$ levels.

The above procedure achieves 2-consistency, since it examines any two variables involved in a constraint. Although this constitutes an advance over simple BT, there are still inconsistencies not detected. To illustrate this, suppose the following form of three domains:

$$\begin{aligned} D_{X_{22}} &= \{2,3\} \\ D_{X_{23}} &= \{2,3\} \\ D_{X_{24}} &= \{2,3\} \end{aligned} \tag{5.14}$$

Since these three variables must possess pairwise different values, the problem at this node is clearly infeasible. This however will not be detected without additional branching, since no domain has been annihilated. The following remark generalises this observation by stating a necessary criterion for the existence of a solution at any node.

Remark 5.1 *Let $S_{i_0} = \{X_{ij} : i = i_0$ and X_{ij} uninstantiated$\}$ and $D_{i_0} = \bigcup\{D_{ij} : X_{ij} \in S_{i_0}\}$, $i_0 = 0,\ldots,n-1$. Similarly, let $S_{j_0} = \{X_{ij} : j = j_0$ and X_{ij} uninstantiated$\}$ and $D_{j_0} = \{\bigcup D_{ij} : X_{ij} \in S_{j_0}\}$, $j_0 = 0,\ldots,n-1$. A necessary condition for the existence of a feasible solution, at any step of the algorithm, is that $|S_{i_0}| = |D_{i_0}|$, for each $i_0 \in I$, $|S_{j_0}| = |D_{j_0}|$, for each $j_0 \in J$.*

The above remark also holds for square Y. Actually, it can be stated in a more general fashion, regarding any *all_different* constraint.

Remark 5.2 *Assume the predicate* all_different(S). *Let $S' = \{x \in S : X$ uninstantiated$\}$ and $D_{S'} = \bigcup\{D_x : x \in S'\}$. A necessary condition for the existence of a feasible solution is that $|S'| \leq |D_{S'}|$.*

Examining the number of uninstantiated variables in all rows and columns, along with reconstructing the union of their domains, requires a considerable effort at each node. We simplify the process by introducing the notion of "*degrees of freedom*" (DOF) for each row (column) and value, as the number of cells that a value can be assigned to in a row (column). For square X, let us name as XR_DF_{ik} the degrees of freedom for value k in row i and, similarly, XC_DF_{jk} the degrees of freedom for value k in column j. Variables YR_DF_{il} and YC_DF_{jl} are defined accordingly. Observe that $XR\_DF_{i_0k_0} = 0$ if and only if either $|S_{i_0}| > |D_{i_0}|$ or value k_0 appears in row i_0. Consider again the case (5.14), and suppose that X_{11}, X_{12}, X_{13} are the only uninstantiated variables of the row 1 of square X. Since the remaining $(n-3)$ cells of row 1 have been instantiated to $(n-3)$ different values, there must exist a value k_0, not appearing in any cell of row 2, such that $XR\_DF_{2k_0} = 0$.

Hence, incorporating degrees of freedom extends the inference generated by constraint propagation. Whenever a domain value k_0 is removed from $D_{X_{i_0j_0}}$, both $XR\_DF_{i_0k_0}$ and $XC\_DF_{j_0k_0}$ are reduced by 1. If a degree of freedom becomes 0, the node is pruned as infeasible. If a degree of freedom for value k_0 becomes 1, the single variable having still k_0 in its domain is found and instantiated to value k_0, independently of which other values exist in its domain. The constraint propagation procedure is, subsequently, invoked. It is easy to see that updating degrees of freedom requires $O(n)$ steps.

The notion of degrees of freedom can be naturally extended to pairs of values. There are n^2 ordered pairs to be assigned and there must exist at least one possible place for each square in order for the problem to remain feasible. We denote as P_DF_{kl} the number of cells, which can still accommodate pair (k,l). To explain the method for updating counters P_DF_{kl}, suppose that value k_0 is removed from $D_{X_{i_0j_0}}$. The pairs that can no longer be assigned to cell (i_0, j_0) because of this last deletion are of the form (k_0, l_0), where $l_0 \in D_{Y_{i_0j_0}}$. Hence, if k_0 is removed from $D_{X_{i_0j_0}}$, all counters $\{P\_DF_{k_0l_0} : l_0 \in D_{Y_{i_0j_0}}\}$ must be reduced by 1. If $P\_DF_{k_0l_0}$ becomes 0 and pair (k_0, l_0) appears in none of the fixed cells, the node is declared infeasible. If $P\_DF_{k_0l_0}$ becomes 1, the single cell (i_1, j_1) that can accommodate pair (k_0, l_0) is fixed, i.e. variables $X_{i_1j_1}$ and $Y_{i_1j_1}$ are instantiated to values k_0 and l_0, respectively. Updating degrees of freedom for pairs is more expensive: if $X_{i_0j_0}$ is set to k_0, k_0 is removed from at most $O(n)$ domains and each removal requires the reduction by 1 of $O(n)$ counters of $P\_DF_{k_0l}$. In total, the process requires $O(n^2)$ steps.

A critical observation is that degrees of freedom for single values and pairs are essentially implicit constraints. These constraints are imposed by the different representations of the problem, as discussed at the end of Section 5.3.1. For example, the degrees of freedom for rows and values of square X, i.e. variables $XR\_DF_{i_0k_0}$, arise from representation (II).

Additional domain filtering can be performed because of the particular structure exhibited by the *all_different* predicate. Assume the following status of domains:

$$\begin{aligned} D_{X_{22}} &= \{2,3\} \\ D_{X_{23}} &= \{2,3\} \\ D_{X_{24}} &= \{2,3,4,5\} \\ D_{X_{25}} &= \{2,3,4,5\} \end{aligned}$$

and let variables $X_{22}, ..., X_{25}$ be the only ones uninstantiated in row 2. This instance is not infeasible, but it is clear that values 2, 3 could never be assigned to variables X_{24}, X_{25}. Removing values 2, 3 from $D_{X_{24}}, D_{X_{25}}$ eliminates certain subproblems and might also have further repercussions, e.g. in terms of degrees of freedom for values 2, 3 in columns 4, 5. Notice that no degree of freedom can imply such a deletion. This case is exclusively captured by the filtering algorithm presented in [77]. This algorithm achieves generalised arc-consistency for a single *all_different* constraint. Its disadvantage is its running time: it requires $O(p^2d^2)$ steps for a constraint on p variables with domains of cardinality at most d. Hence, this propagation procedure needs $O(n^4)$ steps for each of the *all_different* constraints regarding rows and columns of X and Y, i.e. $O(n^5)$ steps in total. It also requires $O(n^8)$ steps for the *all_different* constraint on the Z_{ij} variables in (5.13).

Note that, whenever the filtering is applied to an *all_different* constraint, only uninstantiated variables are considered. This implies that, in contrast to all previous propagation procedures, this scheme performs consistency checks also among uninstantiated variables. Actually, this algorithm accomplishes *hyperarc consistency* for the variable set of an *all_different* constraint ([53, Section 11.3]). A constraint is *hyperarc consistent* when any value remaining in the domain of any variable can be part of a feasible solution. Notice that this does not imply that simply selecting a value from each domain can construct a feasible solution ([53, Section 11.1]).

Each time the algorithm is called for a row/column or for the Z_{ij} variables, certain domain members are deleted. This obviously has further effects, even if a single value is removed, therefore the process must be iteratively called. An upper bound on the number of iterations is mandatory, in order to avoid calling this elaborate scheme only for an unsubstantial amount of domain reduction. A practical option is to set this bound as a multiple of n. The steps of the CP algorithm at each node Q^s are summarised in the following procedure.

Algorithm 5.4

```
repeat
{
(I)     Perform simple domain reduction on $Q^s$;
          if ($Q^s$ is feasible)
(II)            Update degrees of freedom for rows/columns;
          if ($Q^s$ is feasible)
(III)           Update degrees of freedom for pairs;
          if ($Q^s$ is feasible)
(IV)            Apply the filtering algorithm to all_different constraints for variables $X_{ij}$, $Y_{ij}$;
          if ($Q^s$ is feasible)
(V)             Apply the filtering algorithm to all_different constraint for variables $Z_{ij}$ ;
}
until ($Q^s$ is infeasible) or (a solution is found) or (no more domain values are removed)
if ($Q^s$ is still feasible) and (no solution has been found)
     Partition $Q^s$ into subproblems;
else
     prune $Q^s$;
return;
```

Notice the analogy between Algorithm 5.4 and Algorithm 4.1 of Section 4.4.

Being significantly more expensive than any other propagation procedure, this filtering scheme should better be applied in a subset of nodes. The trade-off is obviously between more nodes in the tree and more rigorous domain reduction in each node. Section 5.4 examines this question and provides computational evidence assessing the performance of all propagation schemes. Table 5.2 summarises the various methods for constraint propagation, along with their complexity.

5.3.4 Implementation details

All computer codes have been implemented in the programming environment of Microsoft C++. Although the code is problem-specific, it incorporates all the routines of a typical CP application. In

Table 5.2: Methods for constraint propagation

METHOD	COMPLEXITY
Simple domain reduction	O(n)
DOF for rows/columns and values	O(n)
DOF for pairs of values	O(n^2)
Filtering algorithm of [77]	O(n^5), O(n^8)

particular, there are routines for node creation, constraint propagation and branching. Nodes are retained in a node list, which follows the "*Last-In-First-Out*" discipline, meaning that the last node inserted will be the first considered when backtracking. This implements the depth first strategy. Obviously, book-keeping tasks must be performed for setting up node information. Each node apparently contains information for all variables, domains and degrees of freedom. A node is created each time a variable X_{ij} or Y_{ij} is instantiated because of branching. Hence, the maximum tree level is $2n^2$. Since the search is "depth-first", the node list can contain no more than $2n^2$ nodes. This fact implies that the space complexity of the algorithm is polynomially bounded.

5.4 Computational experience

In this section we examine possible variants of a CP algorithm for the OLS problem. Algorithms BT and FC are combined with four branching strategies and various levels of constraint propagations. The variants are tested on the OLS problem for orders up to 12. We comment on their comparative performance and analyse their behaviour. We focus first on evaluating the branching strategies and afterwards on issues of constraint propagation. Results are illustrated in terms of elapsed time and nodes created in the search tree. Recall that a node is created whenever a variable is instantiated because of branching. More than one variables may be instantiated at a single node only as a result of domain reduction.

5.4.1 Branching strategies

The following experiment applies the simple BT algorithm, without any domain reduction. The four different branching strategies X_R, X_T, XY_R, XY_T are examined. Before presenting their comparative performance, we conduct an experiment for verifying the correctness of the algorithmic approach.

Notice that both X_R and X_T rules create square X completely before considering square Y. Suppose also that we fix in advance only the first row of both X, Y and the first column of X in natural order, the first column of Y remains intact. If the BT algorithm with any of these branching rules searches for all feasible solutions, it will gradually instantiate X to all reduced Latin squares of order n. Hence, we provide a mechanism for counting the number of reduced Latin squares. Having an upper bound of 5×10^4 seconds on the computation time, it has been possible to enumerate all squares up to $n = 7$. The results are illustrated in Table 5.3. The fact that these results coincide with those in the literature ([34, 60]) proves that no feasible solutions are omitted.

Table 5.4 depicts results for the performance of algorithm BT under the four branching rules. All variables and domains are fixed according to Table 4.2 before the search is initiated. Nodes are presented in logarithmic scale with base 2, exactly because their number becomes extremely

Table 5.3: Counting the number of reduced Latin squares

Order of Latin squares	4	5	6	7
Number of reduced Latin squares	4	56	9408	16,942,080

large. An asterisk denotes that no result was achieved within the time limit of 5 × 10^4 seconds. The most critical finding is that no variant of BT algorithm terminates within the time limit for $n \geq 10$. This already illustrates the inefficiency of this approach. In terms of branching rules, X_T and XY_T appear clearly better, with XY_T being slightly more efficient. These branching rules are faster simply because they create fewer nodes, thus verifying once more the analysis of Section 4.4.2. Observe also that rule XY_R is consistently better that X_R. It generally appears that, in the case of fixing rows, it is more beneficial to fix both squares simultaneously. In contrast, when fixing 0s, 1s, etc. it is not remarkably worse to fix square X first. Notice the special case of $n = 8$, where branching rules X_R and XY_R are significantly slower. The reason is that the first feasible solution is a pair of OLS, where square X has only 0s in its main diagonal. It can be seen that this square appears much earlier in the lexicographic order imposed when fixing transversals. Another issue is that the increase in the number of nodes illustrates the exponential explosion with respect to n.

Table 5.4: Comparing branching rules for the BT algorithm

n	TIME				NODES(LOG)			
	X_R	X_T	XY_R	XY_T	X_R	X_T	XY_R	XY_T
4	1	1	1	1	3.6	2.4	3.1	2.4
5	2	2	1	1	9.4	8.7	9.1	8.1
6	11647	8762	8984	8741	20.7	19.6	19.8	18.7
7	16326	14728	15073	14173	24.4	23.3	23.9	22.8
8	30279	1143	28152	1079	26.2	11.3	25.5	10.8
9	44638	42285	45731	40173	29.8	28.9	29.2	287
10	*	*	*	*	*	*	*	*

We conduct the same experiments but with algorithm FC. This algorithm implements only simple domain reduction, without examining degrees of freedom. Table 5.5 summarises the results obtained. Elapsed time is reduced by a factor of at least 4 for all branching methods, only because of this elementary domain reduction. The reduction in terms of nodes is even larger, although the amount of work at each node is now increased. The FC is clearly more successful, thus complementing our initial knowledge from the related literature ([14, 56, 84]). Therefore, it will be the standard algorithm hereafter. The comparative performance of branching rules remains identical, with XY_R arising as the most efficient branching scheme. Observe that elapsed time is not particularly affected by whether the two squares are fixed simultaneously or square X is fixed first. Although it appears more reasonable to assign values to both squares, fixing X alone is much easier and, afterwards, fixing Y becomes trivial because of the orthogonality constraints. The obvious setback is that numerous Latin squares of order n, having no orthogonal mate, are generated. This is the reason why, as n grows, the difference in elapsed time between X_T and XY_T becomes more apparent.

The question to be addressed in the next section is whether more rigorous constraint propagation can improve our results further.

Table 5.5: Comparing branching rules for the FC algorithm

n	TIME				NODES(LOG)			
	X_R	X_T	XY_R	XY_T	X_R	X_T	XY_R	XY_T
4	1	1	1	1	3.1	2.3	2.9	1.5
5	2	2	1	1	7.6	6.4	7.3	6.2
6	1734	1264	1984	958	16.7	20.3	20.9	19.4
7	4581	3924	4376	3417	22.8	21.3	22.4	20.7
8	7682	37	7421	26	25.7	12.4	25.5	11.8
9	18378	9385	18026	9173	28.3	25.4	27.2	23.7
10	29985	26793	28732	24138	30.5	26.9	29.6	25.2
11	46015	37198	42783	33259	32.4	28.1	31.4	27.4
12	*	48367	*	47252	*	30.2	*	28.6

5.4.2 Procedures for Constraint Propagation

All algorithms in this section are variants of the FC scheme with the branching rule XY_T. The difference lies in the amount of constraint propagation performed at each node. Recall from Table 5.2 the complexity of each propagation procedure. Notice also that all propagation methods are essentially special cases of the filtering algorithm for the *all_different* constraint ([77]). However, since this filtering scheme is quite uneconomical, the question is to which extent it achieves significantly more domain reduction than the other procedures. This is the main topic addressed here. We retain the FC algorithm with simple domain reduction and add one feature at a time in order to assess the significance of improvement achieved. Table 5.6 presents the results for four variants. The algorithm named "Simple" replicates the column XY_T of Table 5.5. Essentially this variant does not perform Steps (II)-(V) of Algorithm 5.4. The variants "DOF rows" and "DOF pairs" perform additional consistency checks and variable fixing by updating the degrees of freedom, i.e. they also perform Steps (II) and (III) of Algorithm 5.4. We should clarify that algorithm "DOF pairs" uses degrees of freedom for both values and pairs. Finally, the variant named "Filtering" applies the general filtering algorithm for all constraints, including the one for the Z_{ij} variables, whenever a transversal, ($n-1$ variables in each square) is fixed. The filtering algorithm is applied after all other procedures have been performed and as long as the node is still feasible (as in Algorithm 5.4). In all variants, the overall procedure is recursively performed until *(a)* no more domain reduction is possible or *(b)* a feasible solution is found or *(c)* the node becomes infeasible or *(d)* an upper bound of n iterations is reached.

Table 5.6: Comparing propagation procedures

n	TIME				NODES(LOG)			
	Simple	DOF rows	DOF pairs	Filtering	Simple	DOF rows	DOF pairs	Filtering
4	1	1	1	1	1.5	1	1	1
5	1	1	1	1	6.2	5.8	5.3	4.3
6	958	282	174	43	19.4	17.8	17.3	17.1
7	3417	1174	826	523	20.7	19.1	18.9	18.5
8	26	10	8	3	11.8	9.9	9.7	9.3
9	9173	7203	6938	6745	23.7	20.8	30.5	20.2
10	24138	22132	22052	20935	25.2	24.3	24.1	23.6
11	33259	30672	30341	29423	27.4	26.0	25.7	25.2
12	47252	39436	38819	37161	28.6	27.8	27.6	26.9

The general outcome is that constraint propagation exploits more accurately the structure of the problem, thus identifying more implications and inconsistencies. The remarkable reduction in elapsed time accomplished by incorporating degrees of freedom arises exactly from the implicit constraints enforced by this method. The addition of degrees of freedom for pairs achieves less, but still clear, improvement. Finally, the significant improvement achieved by the filtering algorithm proves that plenty of inconsistencies remain even after introducing the degrees of freedom. The extra effort in each node does not counteract this improvement. It is worth noticing that in all instances, after having propagated via degrees of freedom, the filtering algorithm is usually applied for up to 2 iterations; any additional iterations do not reduce the domains any further.

Hence, it sounds rational to apply the filtering algorithm at even more nodes. We experiment with three different strategies. The first simply incorporates the algorithm whenever a pair is fixed, i.e. at most every two levels. The second uses this scheme every $\frac{n}{2}$ levels, whereas the third whenever an entire transversal has been fixed (as in Table 5.6). Table 5.7 summarises the results. Applying the filtering algorithm whenever a pair is fixed reduces dramatically the number of nodes but is much inferior in terms of elapsed time. For larger values of n this restricts significantly the ability to find a solution. The option of using the algorithm every $\frac{n}{2}$ levels is more moderate, but still worse than the option of using it whenever a transversal is fixed. It appears to create only slightly fewer nodes, despite the additional effort. Overall, it appears that transversals offer a convenient, although problem-specific, criterion for controlling the aggressiveness of constraint propagation.

Table 5.7: Frequency of filtering

n	TIME			NODES(LOG)		
	Pair	Every $\frac{n}{2}$ levels	Transversal	All nodes	Every two levels	Transversal
4	1	1	1	1	1	1
5	2	1	1	2.1	3	4.3
6	614	153	43	13.2	16.7	17.1
7	2287	1279	523	15.6	18.1	18.5
8	68	42	3	6.9	8.8	9.3
9	13873	7930	6745	17.3	18.9	20.2
10	27831	22473	20935	21.7	22.4	23.6
11	36165	31138	29423	23.1	24.6	25.2
12	*	42823	37161	*	26.1	26.9

5.5 Concluding remarks

This chapter introduced the enumeration paradigm of CP and presented its application to the OLS case. A compact CP model has been devised, mainly based on the *all_different* predicate. A critical observation is that this model implies six different representations of the problem, which give rise to implicit constraints. These constraints have been formalised via the notion of "degrees of freedom", which can be checked and updated at low cost. Various versions of the Forward Checking algorithm have been tested. Their differences lie in applying various levels of constraint propagation, including a known algorithm for the *all_different* predicate, and in incorporating different branching rules. The computational results illustrate that CP algorithms, which competently exploit the problem's structure, can efficiently handle the OLS problem, by rapidly enumerating the solution space.

The main difference of the CP algorithms from the B&C algorithm (Chapter 4) is the absence of a relaxation analogous to the LP-relaxation. It is worthwhile to notice that an enormous number of nodes need to be examined for large instances but processing each node is significantly faster. This and additional parallelisms reveal the potential of integrating CP and IP methods. Motivated by our aim to solve the OLS problem, we discuss this topic extensively in the next chapter.

Chapter 6

Integrating Constraint and Integer Programming

In this chapter we present algorithms integrating CP and IP. Although the algorithms are specifically designed for the OLS problem, they encompass general ideas concerning CP/IP integration. The computational results show that algorithms integrating CP and IP behave more competently than traditional CP and IP schemes. It is believed that both the algorithmic ideas and the computational analysis of this chapter might have implications for other combinatorial problems.

IP has, in the main, been developed within the discipline of Operational Research, while CP is an "offspring" of the computer science community, mainly articulated within the field of Artificial Intelligence. Having developed separately, CP and IP often use analogous techniques under a different terminology. This fact might have already become clear from the algorithms exhibited in the previous chapters. The necessity to solve large scale optimisation problems ([40]), together with the revelation of strong links between logic and optimisation ([23]), have stimulated a strong interest in successfully integrating IP and CP. On the other hand, propositional reasoning has been successfully applied to solve open quasigroup problems ([93]), while recent work ([44]) has tested a CP/LP algorithm on the problem of quasigroup completion, i.e. completion of Latin squares, commenting on the comparative superiority of this hybrid scheme. Our work is also motivated by ideas, which apply enumerative and computer based methods to address questions in pure mathematics ([58, 93]).

The rest of this chapter is organised as follows. Section 6.1 reviews developments establishing theoretical links between CP and IP. Section 6.2 discusses ideas for integrating the two methods. The proposed algorithms are described in Section 6.3, while computational analysis is presented in Section 6.4. Finally, Section 6.5 presents a hybrid algorithm for triples of MOLS, along with obtained computational results.

6.1 Theoretical connections between CP & IP

Early ideas establishing connections between logic and optimisation can be found in [54, 85, 86]. An important equivalence is that any logical clause corresponds to a linear inequality on binary variables. For example, the disjunction $\bigvee\limits_{i=1}^{n} x_i$ is equivalent to the inequality $\sum\limits_{i=1}^{n} x_i \geq 1$, $x \in \{0,1\}^n$ ([23, 90]). Hence, any formula in Conjunctive Normal Form (CNF) can be represented by a system of linear inequalities on $0-1$ variables and vice versa. This linear system defines a *set-covering* problem, i.e. an integer program of the form $\{Ax \geq e, x \in \{0,1\}^m\}$, where m is the number of atomic propositions. A CNF formula can be solved by employing a form of resolution called *unit resolution.* This procedure selects a clause with a single literal p (if such exists) and deletes all clauses containing p and all appearances of $\neg p$. In other words, unit resolution simply observes that p must be *true* in any feasible solution. A famous theorem (in [15]) states that unit resolution can prove unsatisfiability for a CNF formula if and only if the LP-relaxation of the corresponding set of linear inequalities is infeasible. This result already indicates that LP-based methods can be applied to problems arising from logic. The inverse is also possible: the resolution algorithm can be applied to the CNF formula arising from a linear constraint set in order to generate resolvent inequalities, also called logic cuts ([46, 51, 53]).

Furthermore, the set of feasible solutions to a logical formula has a minimal representation in terms of the strongest possible clauses. These are the *prime implications*, derived from the original set of clauses by repetitively applying the resolution procedure. A clause is a prime implication if it is not absorbed by any other resolvent clause. Hence, prime implications are maximal with respect to set inclusion, i.e. not dominated. The equivalent concept for IP is that of maximal valid inequalities (see Chapter 2 and [68, p.207]). Exactly as prime implications are derived via resolution after a finite number of steps, maximal valid inequalities are generated after a finite number of steps of the Chvátal-Gomory procedure ([45]). This parallelism is exploited in [23, Section 3.7], where SAT is formulated as a set-covering problem and facets of the associated polyhedron are examined. For infeasible problems, resolution will produce the empty clause as a prime implication (as in Section 5.1.1), whereas the Chvátal-Gomory procedure will produce an inequality of the form $0x \leq -1$. An important statement is the following (in [27]):

> There exists a polynomial $p(N, M)$ such that for any unsatisfiable set S of clauses containing M literals, if the unsatisfiability of S has a cutting plane proof of length N, then it has an extended resolution proof of length $p(N, M)$.

For the notion of extended resolution see [23, Section 3.5.5].

Another pioneering result is the observation that the resolution procedure is closely related to the Fourier-Motzkin algorithm for polyhedral projection (in [85]). Resolution eliminates an atomic proposition and appends additional clauses to the initial representation in a way analogous to that of eliminating variables in the Fourier-Motzkin method. A relevant development concerns the representability of mixed integer linear programming problems (MILP) ([54]). As noted in [23, Section 4.4.3], this work essentially proves that any set representable as an MILP can be also represented in a disjunctive form, i.e. with either/or constraints. It also provides a significant characterisation result for integer programming. The actual theorem states that:

> A set in n-space is MILP representable if and only if it is the union of finitely many polyhedra having the same set of recession directions.

A rather recent direction of research examines constraints of special structure, like the *all_different* predicate, and proposes equivalent representations in the form of linear inequalities. The convex hull of all feasible solutions to the *all_different* predicate is described in [89], while analogous results for cardinality rules are presented in [92].

6.2 Algorithmic approaches combining CP & IP

The combinatorial optimisation problems (COP) targeted by both CP and IP are of the following generic form, hereafter called $COP(x, f, C, D)$:

$$min\{f(x) : x \in C, x \in D\} \tag{6.1}$$

In this formulation, x is the vector of variables, f the objective function, D the external product of variables' domains and C a set of constraints, restricting the possible values that the variables can take simultaneously. The variable domains can be integers, symbols or intervals of real numbers. A *relaxation* of (6.1), called $REL(x, f, C', D')$, is defined as:

$$min\{f(x) : x \in C', x \in D'\} \tag{6.2}$$

where $C' \subseteq C$ and $D' \supseteq D$, i.e. (6.2) is derived by dropping at least one constraint or by enlarging the domain of at least one variable. This implies that the set of feasible solutions of (6.2) includes that of (6.1). Hence, a solution to (6.2) is not necessarily a solution to (6.1). If, however, (6.2) has no solution, neither does (6.1).

Every algorithmic method for (6.1) adopts further assumptions about the form of f, C and D. In particular, IP requires both the objective function and the set of constraints to be linear. This fact highly restricts its declarative power, but allows it to produce efficient problem relaxations. In theory, the form of the constraints in CP is of arbitrary type. However, there is a broadening menu of specific, well-defined, constraint types, which are extensively used in the CP literature, e.g. the *all_different* predicate. CP has been mainly focused on feasibility problems, meaning problems where the set of feasible solutions is relatively small. However, the notion of an objective function can be naturally introduced; hence, CP is adequate also for COP. In general, IP can be efficiently used to solve problems in propositional logic ([23]). Logic, on the other hand, is possible to embed within a classic optimisation framework (see [53]). Both methods have been applied separately to various problems (see [30, 88]). Nonetheless, the integration of the two approaches poses a different task, *viz.* that of using the tools of both CP and IP for modelling and solving COP of general form. The necessity of this direction becomes more apparent as problem types diversify and problem sizes grows.

Chapters 4 & 5 have illustrated that IP & CP models are solved using analogous algorithmic schemes. Both IP and CP use the "Divide & Conquer" principle, i.e. they recursively divide the original problem into subproblems and form a search tree. Both methods are *exact*, in the sense that they guarantee a complete search. The prospect of integration will be discussed with respect to the generic algorithm shown below. Notice that Algorithms 4.1 and 5.4 are special cases of this scheme.

Algorithm 6.1 *At each node/subproblem:*

```
(I)     Preprocess $COP(x, f, C, D)$;
        if (feasible)
        repeat
        {
(II)            Solve $REL(x, f, C', D')$;
(III)           Infer additional constraints;
        }
        until $(x \in D)$ or (infeasible) or (no inference)
    if (no inference) and $(x \notin D)$ and $(REL(x, f, C', D')$ remains feasible))
(IV)     Create subproblems;
    else
        prune this node;
return;
```

Concerning IP, Step I involves variable preprocessing ([78]). Using simple logical connections and information from the variables already fixed, additional variable fixing is attempted. Although not mentioned explicitly in Algorithm 6.1, preprocessing is iteratively applied until no more variables can be fixed (see also Algorithm 4.2). Step II implies solving a continuous relaxation of the problem, derived by dropping the integrality constraints. The *LP relaxation* can be solved efficiently using interior point or simplex-based algorithms. Its main advantage is that it provides an assignment of values to all variables and, if infeasible, it implies infeasibility for the original (sub)problem. If, however, a feasible but non-integer solution is derived, IP proceeds by introducing additional inequalities, which cut-off this fractional solution and restrict further the feasible region D' of the relaxation. This is Step III. The process of solving the relaxation and extracting further inference in the form of cutting planes is repeated until the problem becomes infeasible or an integer solution is found or no additional constraints can be produced. Identifying families of efficient cutting planes normally requires a polyhedral analysis of the problem.

For CP, Steps I & III are essentially the same process. Constraint propagation removes inconsistent values from the domains of the uninstantiated variables. Although domain reduction can be efficiently performed to achieve 1 & 2-consistency, higher consistency levels require considerable computational effort. Exceptions occur in the case of structured constraints, like the *all_different* constraint. Step II is not performed by CP, since it lacks a proper relaxation, although recent work ([53, Chapter 21]) explores the potential of defining useful discrete relaxations. Constraint propagation is repetitively applied until no more domain filtering is possible. The more intense the propagation performed at each node, the larger the computational effort required and the smaller the number of nodes created. This behaviour was observed in Section 5.4 with respect to the OLS problem. The leverage point cannot be theoretically justified and is probably located in moderate options, which apply constraint propagation periodically. The counterpart for IP algorithms is the persistence in generating violated cuts. The number of cuts added to each problem and the frequency of cut addition must balance between efficient solution of subproblems and early pruning of infeasible branches (see Section 4.6). For both methods, the question is identical and is concerned with the effort devoted to infer additional constraints at Step III.

Step IV is conducted in analogous ways by both methods. The standard method is to select a variable and split its domain to create two or more subproblems. Heuristic criteria for this choice are common to both methods. Examples are to select the variable with the smallest domain, or the variable, which participates in most constraints, or the one, which is expected to improve the objective function most. It has been argued ([13]) that branching on Special Ordered Sets is a valuable option for IP. The corresponding branching method for CP is to simply split the domain of the branching variable recursively until a single value remains available. Equivalently, the CP algorithm can directly create a number of nodes equal to the cardinality of the domain and set the branching variable to a different value in each node. Recall that this is exactly the branching mechanism used for the OLS problem.

It follows that CP & IP can be integrated in an algorithmic sense. For Step I, logical preprocessing conducted by IP is a subset of the sophisticated tools available for CP. The crucial contribution of IP is the powerful relaxation lacking from CP. The inference Step (III) can still be performed by both methods. The complementarity exists in the fact that CP directly reduces the solution space of the original problem in contrast to IP, which reduces the solution space of the relaxation. Finally, one of the two, or both, methods can be applied at Step IV to determine the partition of the subproblem.

An important aspect of complementarity is that CP forms only partial assignments of values to variables, except for the point where a feasible solution is identified. In contrast, IP constantly constructs complete assignments for the relaxed problem, which can possibly be feasible solutions for the original problem. Moreover, from a more abstract viewpoint, both methods depend on an interplay between search and inference. Exhaustive search enumerates the possible options for a subproblem, whereas inference accelerates the search by reducing the solution space and by providing certificates of optimality or infeasibility. This fundamental feature is observed and discussed in [52].

Substantial ideas regarding the integration of CP and IP are exhibited in [16], where it is proposed that the constraint set C is partitioned into *primitive* and *non-primitive* constraints. A constraint is characterised as primitive, if an efficient solver is available, which can provide a complete assignment of values to all variables without any further search. The best example is linear constraints on real variables. All other constraints are classified as non-primitive. For example, the only non-primitive constraint in an IP model is an integrality constraint. This work also proposes an interesting algorithmic framework, named "Branch & Infer": at each node of the search tree, both methods infer primitive constraints, e.g. cutting planes, which are added to problem's relaxation. Independently of whether the modelling framework allows for both CP & IP constraints to be inserted, the immediate requirement is for a proper decomposition of the problem. Such a decomposition should allow for an IP and CP solver to be used in parallel, without the overhead of maintaining and updating two models and two search trees becoming overwhelming.

The theoretical connections and equivalences between CP & IP do not guarantee a beneficial integration; they primarily illustrate the feasibility of the project. The virtue of integration can be properly justified only if the inference generated by the two methods can be viewed as complementary. In other words, it has to be concluded that one approach succeeds in cases where the other fails. So far, evidence for this fact remains mostly empirical. In theory, CP is much faster in searching the solution space, since it does not have to solve a relaxation. However, it is exactly the LP-relaxation, which gives IP its global perspective and allows for a solution to be found without extensive branching ([52]). Next section examines, in more detail, algorithms integrating the two methods.

6.3 Design of hybrid algorithms

We propose two algorithms for the OLS problem that incorporate both CP and IP methods. A critical issue concerning the design of these algorithms is whether the problem will be decomposed into two components to be handled by either CP or IP. We select to perform no decomposition, i.e. both CP and IP representations are retained. The first reason is that the cost of maintaining both models is negligible. Most important, we wish to extract the best possible implicit information from both models and also assess the quality of inference achieved by each of the two methods. This will hopefully lead to valuable insights about whether CP and IP complement each other.

According to the above discussion, Steps I-III of Algorithm 6.1 can be performed by CP and IP without any overlap. At each subproblem, Step IV is the only one that has to be determined by a single method. The selected method determines the form of the search tree, since branching via IP always creates two branches, whereas branching on a CP variable results in a number of subproblems equal to the cardinality of its domain. Hence, partition into subproblems can be decided by:

A. IP alone, which implies embedding IP within a CP search tree;

B. CP alone, which implies embedding CP within an IP search tree;

C. both IP and CP, which implies interchanging between the IP and CP search trees according to a certain criterion.

In case (A), Step I is implemented by both IP & CP repetitively. Each method operates on its own model and, if a variable is fixed or a domain is filtered, the information is passed to its counterpart. Steps II-IV are implemented only by IP. Concerning case (B), Steps I & IV are performed by CP alone. IP solver is called at certain nodes and performs the "loop" of steps (II) & (III). The inference achieved by CP in Step (I) is passed to the relaxation in the form of bounds on the (continuous) variables. Finally, case (C) is a combination of the previous two. Both search trees remain active and, whenever the search procedure is transferred from one to the other, all the inference produced is also conveyed. The roles of IP and CP in cases (A) & (B) are reproduced when the search is implemented at the IP or CP tree, respectively.

These cases encompass all options for hybridisation, with case (C) integrating the two methods more closely. The algorithms proposed for the OLS problem adopt options (A) and (B). Option (C) is incorporated in an algorithm for triples of MOLS, to be presented in Section 6.5.

6.3.1 Algorithm IPC

Algorithm IPC is the B&C algorithm, described in Chapter 4, incorporating also constraint propagation as an additional preprocessing step. At each node of the search tree, IP preprocessing is applied first, exactly as illustrated at Algorithm 4.2. If this procedure terminates without proving infeasibility, the CP preprocessor is called, thus attempting to fix additional variables to 0. The aim is not only to enhance the preprocessing of each node, but also to reveal the extent to which CP can be beneficial in terms of detecting infeasible nodes before the LP is solved.

Observe that there exists an *all_different* constraint associated with each of the 6 constraint sets of model $(IP1)$. For example, consider constraint set (4.6):

$$\sum\{x_{ijkl} : k \in K, l \in L\} = 1, \forall i \in I, j \in J$$

which states that exactly one pair (k, l) can be assigned to any cell (i, j). Assume variables W_{ij}, having domains $D_{W_{ij}} = \{0, ..., n^2 - 1\}$. There is a $1 - 1$ correspondence between any ordered pair (k, l) and a value in $D_{W_{ij}}$. Hence, the constraint is:

$$\textit{all\_different}\{W_{ij} : i \in I, j \in J\}$$

Note that this is exactly the constraint on the Z_{ij} variables of the CP model (in Section 5.3.1). The *all_different* predicates for the remaining constraint sets (4.1)-(4.5) are similarly defined. Again, observe that each of these predicates corresponds to one of the 6 representations discussed in Section 5.3.1.

Let C denote the set of n^4 $0-1$ variables and $F^s \subseteq V$ the set of variables already fixed at a certain node Q^s. Define $W^s = \{W_{ij} : x_{ijkl} \in V \backslash F^s$ for some $k \in K, l \in L\}$, i.e. W^s is the set of cells, which have not been assigned a pair. For $W_{ij} \in W^s$, define $D_{W_{ij}} = \{(k+n \cdot l) : x_{ijkl} \in V \backslash F^s\}$, i.e. $D_{W_{ij}}$ is the set of pairs still allowed for cell (i,j). The extra preprocessing step regarding constraint set (2.1) is the application of the filtering algorithm of [77] to the predicate *all_different*(W). This algorithm will either signify that the predicate, and therefore Q^s, is infeasible, or return a list of domain values to be removed as inconsistent. If value $(k+n \cdot l)$ is removed from $D_{W_{ij}}$, variable x_{ijkl} of the IP model is fixed to 0. The same algorithm is then applied to the *all_different* predicates associated with the constraint sets (2.1)-(2.5). When this process is complete, an IP constraint might have been left with a single variable at its left-hand side not fixed to 0. This variable should be explicitly set to 1. For this reason, the IP preprocessing procedure is called again. Clearly, the entire process must be repeated in order to detect any further repercussions of variable fixing. Algorithm 6.2 illustrates all steps in pseudo-code.

Algorithm 6.2 (IPC)

```
    repeat
    {
        Apply Algorithm 4.2 to $Q^s$;
        if ($Q^s$ remains feasible)
            Apply the filtering algorithm to the all_different constraint associated with (4.1);
        if ($Q^s$ remains feasible)
        {
            Fix variables to 0;
            Apply the filtering algorithm to the all_different constraint associated with (4.2);
        }
        ⋮
        if ($Q^s$ remains feasible)
        {
            Fix variables to 0;
            Apply the filtering algorithm to the all_different constraint associated with (4.6);
        }
        Apply Algorithm 4.2 to $Q^s$;
    }
    until ($Q^s$ becomes infeasible) or (an integer solution is found) or (no more variables are
fixed)
  if ($Q^s$ has become infeasible) or (an integer solution has been found)
    prune $Q^s$;
  return;
```

Once more, an upper bound on the number of iterations has to be defined. A multiple of n appears a reasonable choice for the OLS problem. Moreover, although IP preprocessing is implemented in $O(n^4)$ steps, each call to the filtering algorithm requires $O(n^8)$ steps. This implies that the filtering algorithm should be applied only at a subset of nodes. This issue is to be further analysed via computational experimentation in Section 6.4.

Notice that all other methods for constraint propagation, e.g. degrees of freedom, have been ignored. The reason is that these methods are already captured by IP preprocessing, and would also be enforced when the LP is solved. If the domain of a variable $X_{i_0j_0}$ or $Y_{i_0j_0}$ in CP becomes empty, Algorithm 4.2 will detect that all variables in the left-hand side of row (i_0, j_0) of constraints (2.6) are fixed to 0. On the other hand, the LP will be infeasible since constraint (i_0, j_0) of (2.6) is violated. If the domain of a variable $X_{i_0j_0}$ becomes a singleton $\{k_0\}$, only IP variables of the form $x_{i_0j_0k_0l}$ will remain in the left-hand side of row (i_0, j_0) of (2.6). An identical condition occurs concerning degrees of freedom. For example, if $XC\_DF_{j_0k_0}$ becomes 0, the infeasible row will be (j_0, k_0) of (2.3). Exactly because the IP formulation involves four indices, whose roles are indistinguishable, all 6 symmetric representations exhibited in Section 5.3.1 are encompassed. The only domain reduction (variable fixing) not detected by IP preprocessing is the one accomplished exclusively by the filtering algorithm of [77].

6.3.2 Algorithm CPI

The second hybrid scheme, embedding IP within CP, is Algorithm CPI. It is based on the FC algorithm, the additional feature being to call the IP solver at a subset of subproblems. Assume a certain subproblem Q^s in the search tree created by the FC algorithm. As long as this node remains feasible after constraint propagation and there exist still uninstantiated variables, the status of the variables' domains is passed to IP in the form of variable fixing. Clearly, if $k_0 \notin D_{X_{i_0j_0}}$, all variables $\{x_{i_0j_0k_0l} : l \in L\}$ are fixed to 0. Similarly, if $l_0 \notin D_{Y_{i_0j_0}}$, all variables $\{x_{i_0j_0kl_0} : k_0 \in K\}$ are fixed to 0. A variable $x_{i_0j_0k_0l_0}$ is set to 1 only if $X_{i_0j_0} = k_0$ and $Y_{i_0j_0} = l_0$. Recall from Section 4.4.1 that IP variables are fixed by altering their bounds. The process of passing CP information to IP involves checking the domains of all X_{ij} and Y_{ij} variables and requires $O(n^4)$ steps. Before calling the IP solver, all constraint propagation procedures of CP are applied (Table 5.2). The reason is not only to avoid solving an infeasible IP but also to detect cases where IP can prove infeasibility although CP cannot. Integer preprocessing is not performed, since its task has already been accomplished by constraint propagation in CP.

IP is called in order to extend a partial solution achieved by CP to a complete one. By solving the LP-relaxation, IP assigns values to all problem variables. Since fractional solutions might occur, the IP solver generates cutting planes for up to $2 \cdot n$ iterations. The procedure for cut addition is the "Cliques first" scheme, exactly as presented in Section 4.6.4. The initial LP is solved by the Newton-Barrier methods, whereas all other iterations use the Dual Simplex algorithm. A random objective function is introduced in order to avoid degeneracy in the dual LP (see Section 4.4.4). If the procedure is completed without proving infeasibility and without identifying an integer feasible solution, the IP solver is terminated and CP determines the next subproblem to be examined. Hence, the IP algorithm solves a single node without performing any branching. The reason is that our aim is to either extend a partial CP solution to a complete one or prune a branch as infeasible. This approach is also valuable in terms of comparing the inference strength of the two methods.

In fact, a more general framework for integrating IP and CP can be defined. Whereas CP is used to efficiently enumerate a large set of subproblems, IP is periodically called in order to possibly generate a complete assignment early in the search. The valid inequalities of IP complement CP's ability to identify infeasible branches. In theory, this is a closer integration than certain algorithms presented in the literature. For example, the CP/LP algorithms presented in [44, 65] solve only a single LP for deriving reduced costs, which are subsequently used for selecting the most promising branching variable. Apart from the fact that this strategy suits an optimisation problem, hence is not applicable to OLS, it is also a scheme ignoring the inference power of IP that is mainly manifested in the form of cutting planes.

The only issue not discussed is the criterion determining when the IP solver is called. It is easy to see that the IP solver consumes significant computational time, therefore it should not be called at every single node. One the other hand, it is expected that the larger the number of nodes applying IP, the better the inference generated. This question is further analysed in the following section via experimentation.

6.4 Computational experience

This section investigates the performance of algorithms IPC and CPI and compares their behaviour with that of "pure" IP and CP algorithms. All algorithms return a solution or prove that the problem is infeasible for the case of $n = 6$, where a complete search is required. Most important, all algorithms incorporate the preliminary variable fixing, as exhibited at Section 4.2 and apply the same branching mechanism, as first presented in Section 4.4.2. The implementation of IP and CP components has been presented in Sections 4.5 and 5.3.4, respectively. All experiments were conducted on a PC under WinNT, with a Pentium-III processor, 800MHz and 256Mb of main memory.

First, we examine separately the performance of each of the two hybrid algorithms and comment on the aspects influencing their behaviour. The last part of this section discusses the comparative performance of the hybrid algorithms against the IP and CP algorithms of Chapters 4 and 5.

6.4.1 Algorithm IPC

The main question concerning algorithm IPC is the frequency of calling the additional preprocessing routine, implemented by CP. We present three variants of algorithm IPC, which differ only in the criterion for determining whether CP would be incorporated at a certain node. We name these variants Conservative, Moderate and Aggressive. The conservative version employs CP only whenever IP preprocessing detects that the first column of square Y or an entire transversal has been fixed, i.e. whenever $(n-1)$ variables have been set to 1. Notice that this is also the criterion used by the IP algorithm for cut generation. The moderate variant calls CP procedures whenever half a transversal has been fixed. This condition occurs at most every $\frac{(n-1)}{2} \cdot log_2 n$ levels in the IP search tree. Finally, the aggressive version employs CP whenever a variable is set to 1, i.e. at most every $log_2 n$ levels. Notice that numerous other variations are possible. The ones presented here have been selected only as a representative subset, which is sufficient to illustrate the core findings. The actual criterion is, once more, problem specific. A more general criterion would be to apply CP whenever a fraction of IP variables has been fixed. For example, the conservative approach employs CP whenever $\frac{1}{n-1}$ of the variables have been fixed to 1.

Table 6.1 illustrates the results obtained. The percentage of elapsed time devoted to preprocessing (Algorithm 6.2) is also depicted. The "Nodes" field contains only the nodes where the LP had to be solved, i.e. nodes where infeasibility was not detected during preprocessing. The best performance is exhibited by the moderate approach, which creates more nodes than the aggressive one and significantly less than the conservative one. Apparently, the aggressive version employs more rigorous constraint propagation without managing to substantially reduce the number of subproblems. The additional effort required in a large subset of nodes results in the elapsed time growing dramatically for large n. This is because the percentage of time absorbed by preprocessing is immense. For example, considering $n = 6$, the aggressive strategy proves infeasibility in more than double the time required by the moderate approach. On the other hand, for $n = 8$, it cannot reduce at all the number of subproblems. The conservative version actually improves over pure IP (Table 4.20) in terms of elapsed time, whereas the aggressive one is much worse. The more balanced behaviour of the moderate scheme ensures that the cost of extra preprocessing is compensated by solving fewer subproblems. There exists, once more, a critical point regarding the aggressiveness of constraint propagation. Performing more consistency checks among the uninstantiated variables remains beneficial up to this point and becomes detrimental thereafter.

Table 6.1: Results for algorithm IPC

n	TIME							NODES		
	Cons.		Moder.		Aggr.			Cons.	Moder.	Aggr.
	Prep.	Total	Prep.	Total	LP	Total				
4	1.4	4	1.4	4	1.5	5		1	1	1
5	1.6	10	1.7	10	1.9	14		1	1	1
6	1.7	513	2.4	484	3.3	972		2973	2672	2174
7	1.8	3059	2.5	2937	3.8	4852		273	246	216
8	1.4	246	1.8	231	3.2	483		51	47	47
9	2.1	16732	2.7	15826	3.6	22561		32471	30284	29517
10	2.0	21173	2.5	19306	3.8	26285		4.93E+5	3.71E+5	3.08E+5
11	2.2	24834	2.7	23538	3.7	31741		8.37E+5	7.41E+5	6.83E+5
12	2.1	31497	2.9	29213	3.6	48749		2.12E+6	1.84E+6	1.28E+6

6.4.2 Algorithm CPI

The experiments conducted for algorithm CPI are analogous. The issue addressed here is the criterion for calling the IP solver at a CP node. For clarity of presentation, the three variants examined are almost similar to the variants of algorithm IPC. The schemes named Conservative, Moderate and Aggressive, call the IP solver whenever an entire transversal (i.e. $2(n-1)$ CP variables), half a transversal or two pairs (i.e. 4 CP variables), respectively, have been fixed. Computational results are depicted in Table 6.2. Nodes are presented in logarithmic scale.

Not surprisingly, the aggressive approach behaves poorly, regardless of the small number of nodes it creates. Identifying violates cuts and, mostly, re-optimising LPs is an expensive task, as discussed at Section 4.7. Although the reduction in the number of subproblems signifies that cutting planes can detect infeasible subproblems much earlier, the time required becomes prohibitive. Perhaps the most critical observation is that the conservative approach is the best for orders up to 10. This implies that, for smaller problem instances, IP should be called less frequently, even if the number of nodes remains unaffected. However, the picture is inverted for $n = 11$ and 12, where IP's contribution, through the moderate strategy, becomes more substantial. As n grows, cutting planes are more powerful for pruning despite the fact that re-optimising LPs becomes also more laborious.

6.4.3 The benefits of integrating IP and CP

This section replicates results presented previously, in order to examine the impact of integrating IP and CP methods. Table 6.3 summarises the performance of the best variants of the B&C algorithm (Chapter 4) and the FC algorithm (Chapter 5). Hence, columns labelled IP and CP reproduce results from Table 4.19 (Section 4.6.5) and Table 5.6 (Section 5.4.2), respectively. The remaining columns,

Table 6.2: Results for algorithm CPI

n	TIME				NODES (LOG)		
	Cons.	Moder.	Aggr.		Cons.	Moder.	Aggr.
4	1	1	1		1	1	1
5	1	1	1		4.1	3.5	3.4
6	287	376	1056		16.4	14.6	14.2
7	597	854	1636		18.1	16.0	15.7
8	83	112	392		8.9	7.4	6.8
9	12376	14439	17834		19.8	18.4	17.7
10	19481	17842	24507		23.2	21.8	21.2
11	27573	21651	32174		24.8	23.4	22.8
12	34958	26983	39861		26.4	25.1	24.9

namely IPC and CPI, are replicates of tables in the previous sections. Note that the number of nodes is illustrated in logarithmic scale.

For all feasible instances, all algorithms return the same pair of OLS. Table 6.3 primarily shows that all schemes present the same general behaviour, with complexity exploding after $n = 9$. Focusing first on the comparative performance of IP and CP, observe that IP is initially much slower. CP is the best method for the complete search required for $n = 6$. However, the gap is gradually reduced and IP becomes more efficient for $n \geq 11$. Although the burden of solving linear programs becomes larger as n increases, the global view adopted by IP proves to be more effective. For large values of n, CP is practically absorbed in enumerating fruitless branches in the tree, whereas IP can detect those branches much earlier. This is also reflected, and explained, by the substantial difference in the number of nodes. These results resemble the theoretical evaluations and experimental results of related literature (e.g. [30, 52]).

Algorithm IPC is consistently faster than algorithm IP and also creates slightly less nodes. Therefore, incorporating CP within IP is clearly advantageous. The improvement arises from the fact that more subproblems are pruned as infeasible without having to solve the LP-relaxation. Moreover, by fixing even more variables, preprocessing reduces further the size of the LP-matrix, thus accelerating its solution. The extra effort required for the CP procedure does not affect the performance, precisely because it is called periodically. Between two successive calls to CP, domains have been sufficiently altered for the filtering algorithm of [77] to detect a notable number of variables that must be set to 0. In this way, CP is capable of compensating by generating inference, which would be intractable otherwise.

The improvement achieved in CPI by employing IP within CP is more remarkable. Observe first the striking difference in the number of nodes created by CP and CPI. This constitutes a clear indication of CPI pruning infeasible branches much earlier by complementing CP's inference strength with that of IP. In terms of elapsed time, CPI is slower than CP for orders up to 10 and eminently better thereafter.

Overall, incorporating CP within IP is indeed beneficial but does not really alter the differences between the performance of CP and IP. In particular, algorithm IPC behaves in approximately the same fashion as algorithm IP and is worse than CP for small orders. A much more balanced and robust performance is accomplished by the second hybrid scheme. In terms of both time and nodes, CPI lies approximately in the "middle" of IP and CP for orders up to 9. For $n \geq 9$, CPI outperforms all other schemes. Hence, this scheme appears to integrate more efficiently CP and IP.

Table 6.4 provides further insights on the performance of the hybrid algorithms. It depicts the percentage of infeasible nodes pruned by IP in algorithm CPI and the percentage of infeasible nodes

Table 6.3: Comparative results for CP/IP algorithms

n	TIME					NODES (LOG)			
	IP	IPC	CP	CPI		IP	IPC	CP	CPI
4	4	4	1	1		0.0	0.0	1.0	1.0
5	11	10	1	1		0.0	0.0	4.3	3.58
6	527	484	43	376		11.5	11.4	17.1	14.65
7	3278	2937	523	854		8.1	7.9	18.5	16.0
8	254	231	3	112		5.8	5.6	9.3	7.4
9	17592	15826	6745	14439		15.0	14.9	20.2	18.4
10	21903	19306	20935	17842		18.6	18.5	23.6	21.8
11	25713	23538	29423	21651		19.6	19.5	25.2	23.4
12	33872	29213	37161	26983		21.2	20.8	26.9	25.1

pruned by CP in algorithm IPC. In more detail, indicator INFC is the percentage of infeasible nodes in the search tree of IPC, which were detected by CP during preprocessing. On the other hand, INFI denotes the percentage of infeasible nodes in the search tree of CPI, which were pruned only by solving the corresponding IP. Recall that these nodes correspond to subproblems made already consistent by CP alone. Therefore, INFI also indicates the percentage of cases where the inference generated by IP could achieve what CP alone could not, except by further branching. In other words, INFC and INFI are indicators of the usefulness of incorporating CP within the IP search tree and IP within the CP search tree, respectively. According to Table 6.4, only up to 6% of the nodes in the IP search tree are pruned as infeasible during preprocessing. On the contrary, at least 25% of nodes in the CP search tree are pruned by IP. Note that for $n = 6$, INFI rises to 66%. This explains the superior performance of algorithm CPI in terms of nodes. CP is clearly useful within IP but to a smaller extent.

Table 6.4: Percentage of infeasible nodes pruned

n	5	6	7	8	9	10	11	12
INFI	0.532	0.657	0.416	0.227	0.321	0.315	0.264	0.250
INFC	0.0	0.046	0.052	0.059	0.032	0.032	0.048	0.024

Another measure of the inference strength is the early identification of infeasible branches. A representative criterion, independent of the actual form of the search tree, is the number of transversals fixed before pruning a node. We briefly report the results regarding the infeasible case of $n = 6$. Algorithms IP, IPC and CPI need to fix at most one transversal before proving that a specific branch is infeasible, i.e. all of them generate approximately the same quality of inference. In contrast, algorithm CP is able to prune a node as infeasible, after fixing a single transversal, only in around 10% of the cases; in the remaining ones, it has to fix up to 4 transversals before proving infeasibility. Analogous results appear for larger values of n and explain the fact that algorithm CP creates the largest number of nodes. Algorithm CPI is faster than algorithms IP and IPC for $n = 6$, exactly because it solves a much smaller number of linear programs and employs IP only after the problem has been restricted enough for cutting planes to become capable of proving infeasibility. On the contrary, algorithms IP and IPC insist on solving LPs and generating cutting planes in between without pruning any further branches.

These observations have obvious implications for more general combinatorial optimisation problems. Given that a problem is in $\mathcal{NP}$, obtaining a complete polyhedral description is as hopeless as finding a polynomial algorithm. Partial polyhedral description can be efficiently used to reduce the search, but usually cannot become effective unless the problem has been sufficiently restricted. For example, obtaining an integer solution or a certificate of infeasibility at the top node of a B&C tree is possible only for very small problem instances. For larger instances, CP can provide substantial improvement by enumerating all partial assignments to a certain number of variables. IP can then be applied to the restricted problem and attempt to extend the partial solution to a complete one or prune the partial solution without any further enumeration. If IP fails to do either, CP can again be used to extend the partial solution by instantiating further variables before IP is called again. The point where the algorithm must "switch" between CP and IP is, most probably, problem specific. In the case of OLS, the notion of transversals offers a convenient criterion. Finally, the role of a useful relaxation appears to be crucial, since inference is more efficiently generated for the relaxed rather than the original problem.

6.5 An algorithm for triples of MOLS

This section extends the ideas presented above to the problem of identifying a triple of Mutually Orthogonal Latin Squares (MOLS) of order n. Recall from Section 1.1 that k Latin Squares ($k \geq 3$) are called *mutually orthogonal* if they are pairwise orthogonal. It is known that there can exist at most $n-1$ MOLS of order n. This upper bound is actually attainable if n is a prime power and for all $n \leq 9$, except of course for $n = 6$. For all other n, the maximum number of MOLS remains unknown. The lack of a general theoretical argument has motivated the use of computer-based methods to answer the question for small values of n. The most famous example is the proof for the non-existence of a set of 9 MOLS of order 10 ([58]).

The case of $n = 10$ is particularly interesting because it is the smallest order, which disproves Euler's conjecture. Recall that Euler had conjectured the non-existence of a pair of OLS when n is an odd multiple of 2. The conjecture is true only for $n = 2, 6$ and fails for the first time for $n = 10$. Although it has been easy to construct a pair of OLS of order 10, multiple attempts have failed to answer the question about whether any such pair is extendible to a triple. It follows that 10 is the smallest order for which the maximum number of MOLS remains unknown.

Based on the experience accumulated for the OLS problem, we design an algorithm that identifies triples of MOLS. This algorithm is integrating IP and CP methods, motivated by the results of the previous sections. The algorithm is successful for orders up to 9 and fails to answer the question for $n = 10$. Note that if there exists no triple of MOLS for $n = 10$, a complete enumeration is required. Since this last instance appears intractable, we descend to only providing an upper bound on the computation time, in case the problem is indeed infeasible. The related literature contains numerous construction methods for sets of MOLS, maximal or not (see [35, Chapter 5]). These methods are straightforward, in the sense that no search is required. However, they are valid only if n has specific properties, e.g. prime number. Although our method is enumerative and, in the worst case, of exponential complexity, it appears to be more generic in the sense of employing no further assumptions about the value of n. This is probably a fundamental discourse between algorithmic and analytical methods.

We proceed by exhibiting IP and CP models for the problem of a triple of MOLS, or 3-MOLS for short. Next, we present the algorithm and discuss computational results.

6.5.1 IP and CP models for the 3-MOLS problem

The models of this section anticipate the more general models presented in the next chapter. Let X, Y, U denote the three Latin squares of order n, which are pairwise orthogonal.

Considering first the IP model, assume five n-sets, namely I, J, K, L, M, indexing the rows, columns and values of the three squares, respectively. Let the binary variable x_{ijklm} be 1, if cell (i, j) of square X, Y, U contains value k, l, m, respectively. Since squares Y and U must be orthogonal, it follows that each ordered pair (l, m) must occur exactly once, regardless of value k. The constraint ensuring this property is:

$$\sum\{x_{ijklm} : i \in I, j \in J, k \in K\} = 1, \forall l \in L, m \in M \tag{6.3}$$

Exactly as in Section 1.5, by taking into account that the roles of the five sets are purely conventional, we are allowed to interchange them. This leads to $\binom{5}{2} = 10$ constraint types, analogous to (6.3). The sum of all variables defines the objective function. Hence, the IP model for the 3-MOLS problem is the following.

$$\begin{aligned}
& \max \sum\{x_{ijklm} : i \in I, j \in J, k \in K, L \in L, m \in M\} \\
\text{subject to} \quad : \quad & \\
& \sum\{x_{ijklm} : i \in I, j \in J, k \in K\} = 1, \forall l \in L, m \in M \\
& \sum\{x_{ijklm} : i \in I, j \in J, l \in L\} = 1, \forall k \in K, m \in M \\
& \sum\{x_{ijklm} : i \in I, j \in J, m \in M\} = 1, \forall k \in K, l \in L \\
& \sum\{x_{ijklm} : i \in I, k \in K, l \in L\} = 1, \forall j \in J, m \in M \\
& \vdots \\
& \sum\{x_{ijklm} : k \in K, l \in L, m \in M\} = 1, \forall i \in I, j \in J \\
& x_{ijklm} \in \{0, 1\} \forall i \in I, j \in J, k \in K, l \in L, m \in M
\end{aligned} \tag{6.4}$$

This model is defined on n^5 $0-1$ variables and contains $10n^2$ equality constraints, each involving n^3 variables. Each integer feasible solution has exactly n^2 variables set to 1, one for each cell (i, j).

The CP model is more intuitive. Let variables X_{ij}, Y_{ij}, U_{ij} denote the content of cell (i, j) in squares X, Y, U, respectively. The domain for all these variables is $\{0, ..., n-1\}$. As in Section 5.3.1, auxilliary variables $Z_{ij}^{xy}, Z_{ij}^{xu}, Z_{ij}^{yu}$ are introduced in order to enforce the orthogonality condition between each of the three pairs of OLS. The domain of each Z variable is $\{0, ..., n^2-1\}$, where each value has a $1-1$ correspondence with one of the ordered pairs $\{(0,0), ..., (n-1, n-1)\}$. It is not difficult to see that the CP formulation for 3-MOLS is the following.

$$\begin{aligned}
\mathit{all\_different}\{X_{ij} \; &: \; i \in I\}, \text{ for all } j \in J \\
\mathit{all\_different}\{X_{ij} \; &: \; j \in J\}, \text{ for all } i \in I \\
\mathit{all\_different}\{Y_{ij} \; &: \; i \in I\}, \text{ for all } j \in J \\
\mathit{all\_different}\{Y_{ij} \; &: \; j \in J\}, \text{ for all } i \in I
\end{aligned}$$

$$\begin{aligned}
&\mathit{all\_different}\{U_{ij} : i \in I\}, \text{ for all } j \in J\\
&\mathit{all\_different}\{U_{ij} : j \in J\}, \text{ for all } i \in I\\
&\mathit{all\_different}\{Z_{ij}^{xy} : i \in I, j \in J\}\\
&\mathit{all\_differen}\{Z_{ij}^{xu} : i \in I, j \in J\}\\
&\mathit{all\_different}\{Z_{ij}^{yu} : i \in I, j \in J\}\\
&Z_{ij}^{xy} = X_{ij} + n \cdot Y_{ij}, \text{ for all } i \in I, j \in J\\
&Z_{ij}^{xu} = X_{ij} + n \cdot U_{ij}, \text{ for all } i \in I, j \in J\\
&Z_{ij}^{yu} = Y_{ij} + n \cdot U_{ij}, \text{ for all } i \in I, j \in J\\
&D_X = D_Y = D_U = \{0, \ldots, n-1\},\\
&D_{Z^{xy}} = D_{Z^{xu}} = D_{Z^{yu}} = \{0, \ldots, n^2-1\}
\end{aligned} \tag{6.5}$$

This formulation requires $6n^2$ variables, $6n$ *all_different* predicates of cardinality n, 3 *all_different* predicates of cardinality n^2 and $3n^2$ equality constraints. It is clearly a more compact representation of the problem, involving significantly less variables that the IP model.

6.5.2 Algorithms for the 3-MOLS problem

It is easy to see that the size of the IP model is enormous. Therefore a pure IP algorithm could not constitute a viable solution method, except for small values of n. For $n = 10$, this model would include 10^5 binary variables and even a single linear program would take several minutes to be solved. On the other hand, the CP model is not prohibitively large. Hence, a CP algorithm, analogous to the one of Chapter 5, is applied to the 3-MOLS problem, mainly for assessment purposes. Nevertheless, the superiority of algorithm CPI in large instances of the OLS problems indicates that this algorithm should also be the most appropriate for the 3-MOLS problem. Especially if the instance of $n = 10$ is infeasible, it is of crucial importance that the best possible inference is extracted by both CP and IP.

Before presenting the algorithms, we must note that the preliminary variable fixing of Table 4.2 (Section 4.2) is naturally extended to triples of OLS. It is not difficult to see that the first row in all three squares can be fixed in natural order. Similarly, the first column of squares X and Y is fixed as in Table 4.2, whereas the first column of square U remains intact. The only values removed from the domains of variables $U_{i0}, i \in I$, are the ones excluded by the orthogonality constraints. For example, since $X_{10} = 1, Y_{10} = 2$, values $1, 2$ are initially removed from the domain of variable U_{10}.

The CP algorithm incorporates model (6.5) and is a simple extension of the algorithm described in Section 5.3.3. At each step, the algorithm selects a cell (i, j) for which X_{ij} is uninstantiated. It then instantiates variable X_{ij} first, followed by variables Y_{ij} and U_{ij}. The algorithm always selects a cell in a different row and column, thus fixing transversals of squares Y and U. For each such transversal, it is easy to see that all cells in square X are instantiated to the same value, i.e. the algorithm fixes all 0s, then all 1s and so on in square X. Fixing transversals is more efficient than simply fixing rows or columns of the three squares, according to arguments similar to those of Section 4.4.2. At each node of the search tree, the CP algorithm performs simple domain reduction and updates degrees of freedom (DOF) for rows and columns of all three squares. It also checks DOF for pairs of values regarding all three pairs. The filtering algorithm for *all_different* constraints is implemented whenever a transversal, i.e. $(n-1)$ cells in all squares, has been fixed. Notice that the complexity of all procedures for domain reduction is identical to the one presented in Table 5.2 (Section 5.3.3). For example, updating DOF for rows and columns still requires $O(n)$ steps. This constitutes an obvious advantage for CP and arises from the fact that the CP model requires $O(n^2)$ variables for both the OLS and 3-MOLS problems.

The hybrid algorithm, called CPI(3) hereafter, incorporates the main aspects of algorithm CPI. Because of the IP model being immense, it is difficult to maintain both CP and IP models, as for the CPI algorithm. In contrast, the 3-MOLS problem is decomposed into two parts to be handled by CP and IP. In particular, squares X and Y are instantiated using IP, while CP deals only with square U. The search is initiated by CP, which fixes values $0, ..., t$ in square U, where $t \leq n-1$ is a parameter of the algorithm. Each value is assigned to $n-1$ cells in pairwise different rows and columns. Let us define $T_{m_0} = \{(i,j) : U_{ij} = m_0)$ for $m_0 \in \{0,...,t\}$. It is not difficult to see that cells in each T_{m_0} must form a transversal in squares X and Y. The CP algorithm performs domain reduction based only on the constraints concerning square U being Latin. Once values $0,...,t$ have been assigned, the IP solver is called in order to create squares X and Y. These squares must be orthogonal to each other and also to the partially instantiated square U. To accomplish this, further constraints must be appended to the IP model. For each m_0, cells in T_{m_0} must possess pairwise different values. In other words, for every $k_0 \in K$, at most one of the variables $\{x_{ijk_0l} : (i,j) \in T_{m_0}, l \in L\}$ must be equal to 1. Similarly, for every $l_0 \in L$, at most one of the variables $\{x_{ijkl_0} : (i,j) \in T_{m_0}, k \in K\}$ must be equal to 1. Therefore, the additional constraints for the IP model are:

$$\begin{aligned} &\sum\{x_{ijk_0l} &:& \quad (i,j) \in T_{m_0}, l \in L\} = 1, \text{ for all } k_0 \in K, \text{ for all } m_0 \leq t \\ &\sum\{x_{ijkl_0} &:& \quad (i,j) \in T_{m_0}, k \in K\} = 1, \text{ for all } l_0 \in L, \text{ for all } m_0 \leq t \end{aligned} \tag{6.6}$$

This is a total of $2(t+1)n$ constraints. The larger the value of t the more constrained the IP model. For $t = n-1$, i.e. when square U is completely fixed, the number of constraints (6.6) rises to $2n^2$.

The IP model is solved by the Branch & Cut algorithm described in Section 4.4. The outcome is either a feasible solution or a proof of infeasibility. If a feasible integer solution is returned, meaning a pair (X, Y), the CP solver instantiates the remaining cells of square U, under the additional constraints for orthogonality with squares X, Y. If it succeeds, a solution for the overall problem has been found and the algorithm terminates. If not, the IP solver is called again. Note that the partially instantiated square U has not changed, therefore the search in the IP tree must be resumed at the point where it previously stopped. The IP solver might return no feasible solution because either *(a)* there exists no OLS pair (X, Y), orthogonal to partial square U, or *(b)* all feasible solutions have already been generated. In both cases, the CP algorithm has to backtrack and provide another partial square U. In essence, algorithm CPI(3) interchanges between the two solvers and keeps both search trees active. Notice that both CP and IP control the creation of subproblems in different stages, i.e. Step IV of Algorithm 6.1. Hence, this hybrid scheme implements option (C), as described in Section 6.3.

The critical aspect controlling the performance of CPI(3) is parameter t. The larger this parameter, the more cases will be examined by CP for partially generating square U and the more restricted the solution space for the IP solver will be. In general, we would like the IP search tree to remain small, since an IP node requires much more effort than a CP node. This automatically suggests setting t to a large value. However, IP has proved to be more efficient in pruning infeasible branches. Hence, large values of t imply more calls to IP, which do not necessarily exploit IP's inference strength. The extreme case of setting $t = n-1$, which implies examining every single Latin square of order n, is clearly impractical. On the other hand, setting $t = 1$ implies that most of the search will be implemented in the IP search tree. Whether this is indeed efficient is highly correlated with the ability of IP to identify a solution or prove infeasibility early in the search. For the 3-MOLS problem, the question is further addressed via experimentation.

6.5.3 Computational Results

This section provides results for algorithm CPI(3) in terms of elapsed time. It is basically examined whether this algorithm can efficiently identify triples of MOLS and, most important, whether it can search the entire space for $n = 10$ in reasonable time.

First, we compare algorithm CPI(3) with the CP algorithm. Parameter t is set to $\lfloor \frac{n}{2} \rfloor$. Table 6.5 illustrates the elapsed time for orders up to 9. The upper bound on computation time is again 5×10^4 seconds. All algorithms exceed this limit for $n = 10$. Note that the case of $n = 6$ is clearly not applicable. Once more, the results demonstrate the comparative efficiency of the hybrid algorithm. For both $n = 7$ and $n = 8$, algorithm CPI(3) identifies a solution much earlier because the cutting planes of the IP solver force the dual simplex algorithm to select an integer extreme point.

Table 6.5: Identifying triples of MOLS of order n

	Time					
n	4	5	6	7	8	9
CPI(3)	2	12	n/a	1785	4967	18413
CP	1	2	n/a	4729	15370	38194

The next experiment investigates the impact of parameter t on the performance of algorithm CPI(3). The values of t examined are $\{1, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor, \lfloor \frac{3n}{4} \rfloor, (n-1)\}$. The results in terms of elapsed time are depicted in Table 6.6. Again, no algorithm terminates within the time limit for $n = 10$. For $t = (n-1)$, the time limit is also exceeded for $n = 9$. These results show that the extreme values of t constitute inadequate options. For small n, values $\lfloor \frac{n}{4} \rfloor$ and $\lfloor \frac{3n}{4} \rfloor$ are more efficient, also because they are closer to value $\lfloor \frac{n}{2} \rfloor$. For $n = 8$, the best result is achieved for $t = \lfloor \frac{n}{4} \rfloor$. Actually, the optimal value for t is $\lfloor \frac{n}{4} \rfloor + 1 = 3$, although not listed in Table 6.6. As n becomes larger, elapsed time for $t = \lfloor \frac{n}{2} \rfloor$ is significantly shorter. The overall conclusion is that intermediate values of t balance more efficiently the strengths and weaknesses of the two solvers: CP enumerates quickly a number of cases, restricting the problem sufficiently for IP to find a pair (X, Y) in reasonable time.

Table 6.6: Examining the role of parameter t

	Time					
n	4	5	6	7	8	9
$t = 1$	2	14	n/a	2392	11837	34581
$t = \lfloor \frac{n}{4} \rfloor$	2	14	n/a	2392	4146	21068
$t = \lfloor \frac{n}{2} \rfloor$	2	12	n/a	1785	4962	18413
$t = \lfloor \frac{3n}{4} \rfloor$	2	17	n/a	1963	13325	30752
$t = (n-1)$	2	21	n/a	4258	27613	*

Since no result is accomplishable for $n = 10$, we propose a method for deriving an upper bound on the completion time. Since the CP solver is partially fixing square U, it will start by fixing value 0 in $(n-1)$ cells of pairwise different rows and columns. Recall that, by having fixed the first row of U in natural order, value 0 has already appeared in column 0. Hence, there are $(n-1)$ possible cells of row 1, where value 0 can be fixed. Having fixed value 0 in row 1, there exist $(n-2)$ cells of row 2 still having value 0 in their domains. In general, having already fixed 0 in rows $1, ..., i-1$, $(n-i)$ options remain for row i. It is not difficult to see that the total number of ways for fixing 0 in $(n-1)$ cells of U is exactly $F_n = (n-1)!$. For $n = 10$, this number is $F_{10} = 9!$ or $F_{10} \simeq 3.63 \times 10^6$. Let $Time_0$ denote the elapsed time for proving that, for one particular fixing of 0s in U, there exists no solution to 3-MOLS. It follows that $Time_0 \cdot F_{10}$ provides an estimate for the total completion time, if the problem is infeasible.

To check the existence of a solution for a particular fixing of 0s, it is sufficient to include this in the preliminary variable fixing, which is performed before the search starts. This implies that the search will now start by fixing 1s in square U, for each of the F_{10} subproblems. Note that there exists no justifiable argument about whether the solution time of a subproblem varies substantially for different fixings. Therefore, we generate 10 different fixings randomly, run algorithm CPI(3) for each instance and calculate the average elapsed time. The estimated elapsed time, for different values of t, is depicted in Table 6.7.

Observe that the average time is reduced as t increases up to 6 and starts increasing drastically thereafter. Once more, values of t close to $\lfloor \frac{n}{2} \rfloor$ achieve the best results. However, the shortest completion time, in case that there exists no triple of MOLS of order 10, is $(23,724 \text{ seconds}) \cdot F_{10} \simeq 273$ years. Moreover, this is only an estimate, which implies that it is not necessarily accurate. In order to assess the validity of our estimate, we have repeated the experiment for 100 randomly generated fixings of 0s in square U and for $t = 6$. The average time for each instance has been 23,296 seconds, which is relatively close to the average time obtained for 10 instances.

Table 6.7: Estimating the average elapsed time for an instance of $n = 10$

t	1	2	3	4	5	6	7	8	9	10
$Time_0$ (sec)	40573	36827	29634	27613	24867	23724	32159	37492	45091	45134

6.6 Concluding remarks

This chapter has investigated the potential for integrating IP with CP. Certain methods of integrating the two methods have been implemented in the form of hybrid algorithms. Extensive experimentation has been conducted in order to compare these hybrid schemes against the IP and CP algorithms presented in Chapters 4 and 5, respectively, The main conclusion is that hybrid algorithms are more efficient, especially for large problem instances, suggesting that the integration of the two methods is advantageous. Although the experimental results concern exclusively the OLS problem, the observed behaviour allows for conclusions to be drawn for general combinatorial problems. Despite the fact that a hybrid algorithm performs much better for the OLS problem, it has been unable to provide a definite answer to the open question of the existence of a triple of MOLS for $n = 10$. Nevertheless, it is still encouraging that the hybrid algorithm provides much better results, compared to a traditional CP scheme.

Chapter 7

Multidimensional Assignment Problems

This chapter generalises the results, derived for the OLS problem, to multidimensional assignment problems. Assignment structures are embedded in numerous combinatorial optimisation problems. An assignment occurs whenever a member of an entity must be allocated/mapped to a member of another entity. The simplest case of an assignment problem is the well-known 2-index assignment, which is equivalent to the weighted bipartite matching problem. Applications include the assignment of facilities to locations and the assignment of delivery points to vehicles.

Extensions of the assignment structure to more than two entities give rise to *multidimensional* (or *multi-index*) *assignment problems*, formally introduced in [70, 71]. These problems essentially ask for a minimum weight clique partition of the complete k-partite graph (see also [12]). A k-index assignment problem is defined on k sets, usually (but not always) assumed to be of the same cardinality n. The goal is to identify a minimum weight collection of n disjoint tuples, each including a single element from each set. This is the class of *axial* assignment problems ([9, 79]). Its IP formulation involves $0-1$ variables with k indices and the constraint sets are derived by summing over all possible sets of $k-1$ indices. The weighted sum of the variables constitutes the objective function, a fact that justifies the alternative term *linear sum* assignment problems ([22]). As an example, recall the formulation of the 3-index assignment problem ($3AAP_n$), presented at Section 2.1.

If the aim is, instead, to identify a collection of n^2 tuples, partitioned into n disjoint sets of n disjoint tuples, a different structure appears. By way of illustration, consider the problem of allocating n teachers to n student groups for sessions in one of n classrooms and using one of n laboratory facilities (i.e. n^2 quadruples), in such a way that all teachers teach all groups, using each time a different classroom or a different facility (for a relevant case, see [39]). These assignment problems are called *planar* and are directly linked to *Mutually Orthogonal Latin Squares (MOLS)* ([60]). The IP formulation of planar problems can be defined on n^k $0-1$ variables (each having k indices) and the constraint sets are derived by summing over all possible sets of $k-2$ indices. Examples are the formulations of the Latin square problem ($3PAP_n$) and the OLS problem ($4PAP_n$), presented at Section 2.1.

Generalising this concept, we could ask for a minimum weight collection of n^s tuples, thus defining the (k,s) assignment problem of order n, hereafter denoted as $(k,s)AP_n$. According to this notation, $(k,1)\text{AP}_n$ and $(k,2)\text{AP}_n$ denote the k-index axial and planar problems, respectively. A more formal definition appears in the following section. The IP formulation of $(k,s)AP_n$ requires n^k $0{-}1$ variables. Its constraints are derived by summing over all possible sets of $k-s$ out of k indices.

An excellent review for assignment problems is [79], where complexity and approximability issues are covered. It is known that the 2-index problem is polynomially solvable, whereas the 3-index axial and planar problems are $\mathcal{NP}$-hard (see [43] and [55], respectively). The authors of [21] prove that the multidimensional (axial) assignment problem remains $\mathcal{NP}$-hard for $k \geq 4$, if the coefficient vector of the objective function fulfills an *Anti-Monge* condition.

Apart from its theoretical significance, the $(k,s)AP_n$ has interesting applications. Multidimensional (axial) assignment structures have recently received substantial attention, because of their applicability in problems of data association. Such problems arise naturally in multi-target/multi-sensor tracking in satellite surveillance systems (see [72]). An application to the problem of tracking elementary particles at the large electron-positron collider of CERN is reported in [75]. The planar problems share the diverse application fields of *MOLS*, discussed in Section 1.4.

This chapter provides an Integer Programming formulation of the $(k,s)AP_n$ (Section 7.1) for general values of k,s,n. The framework introduced enables the polyhedral study of entire families of assignment problems. A number of boundary conditions is exhibited in Section 7.2.1. The dimension of the linear assignment polytope is established in Section 7.3. In Section 7.4, we examine certain properties of the convex hull of integer vectors, provide a necessary condition for the existence of a solution to the $(k,s)AP_n$ and propose a hierarchy of assignment problems. The dimension of the axial assignment polytopes is established in Section 7.5, while a family of facets is exhibited in Section 7.6. Section 7.7 establishes the dimension of the planar assignment polytopes. Finally, modelling the $(k,s)AP_n$ via Constraint Programming is the topic of Section 7.8.

7.1 Mathematical formulation

This section provides a formal definition of the $(k,s)AP_n$. As mentioned before, we consider an IP model derived by summing over all possible subsets of $k-s$ out of k indices. Such a formulation unifies and generalises known IP models for assignment problems. In order to exhibit this compact formulation, we need to formally define the associated problem, i.e. the $(k,s)AP_n$, which encompasses the definitions of axial and planar assignment problems.

Let us motivate our discussion with a number of examples. Table 7.1 exhibits a pair of OLS of order 4. The sets of rows and columns correspond to student groups and classrooms, while the sets of values for the two squares denote teachers and topics. The $(2,1)AP_4$ asks for an allocation of teachers to groups. A solution consists of 4 ordered pairs or 2-tuples, e.g. $\{(1,1),(2,2),(3,3),(4,4)\}$. The $(3,1)AP_4$ is the problem of allocating each teacher to a single group and a single classroom. A solution is composed of 4 3-tuples of the form (group,classroom,teacher). The solution $\{(0,0,0)$, $(1,1,3)$, $(2,2,1)$, $(3,3,2)\}$ is illustrated by the bordered cells of the first square. Evidently, a solution to $(3,1)\mathrm{AP}_n$ defines a transversal of a Latin square of order n.

If all teachers must teach all groups in all classrooms, the underlying problem is the $(3,2)AP_4$. It follows that each (group,classroom) tuple must be included exactly once and each teacher must be allocated to a set of 4 disjoint 2-tuples. For example, the solution indicated by the first Latin square implies that teacher 0 is allocated to the pairs $\{(0,0)$, $(1,2)$, $(2,3)$, $(3,1)\}$. Hence, the $(3,2)\mathrm{AP}_n$ asks for n^2 2-tuples, which can be partitioned into n disjoint sets of n disjoint 2-tuples. The $(4,2)AP_4$ includes the additional requirement that all topics should be taught to all groups

Table 7.1: Examples of assignment structures

TEACHERS

		Classrooms			
		0	1	2	3
Groups	0	0	1	2	3
	1	2	3	0	1
	2	3	2	1	0
	3	1	0	3	2

TOPICS

		Classrooms			
		0	1	2	3
Groups	0	0	1	2	3
	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0

and in all classrooms. It is not difficult to see that each topic must be associated with 4 disjoint (group,classroom,teacher) tuples. It follows that a solution to $(4,2)\text{AP}_n$ is a set of n^2 3-tuples, which can be partitioned into n disjoint sets of n disjoint 3-tuples.

In general, a solution to the $(k,1)\text{AP}_n$ is composed of n disjoint k-tuples, whereas a solution to the $(k,2)\text{AP}_n$ contains n^2 $(k-1)$-tuples, partitioned into n disjoint sets of n disjoint tuples.

Definition 7.1 *Let $K = \{1,\ldots,k\}$ and an integer $s \in \{1,...,k-1\}$. Consider k disjoint n-sets $M_1, M_2, \ldots, M_k$ and select any $(k-s+1)$ of these sets. Let $i_1,...,i_{k-s+1}$ be any $k-s$ distinct integers from 1 to k. Define the following sequence of powersets:*

$W_0 = \{(m^{i_1},...,m^{i_{k-s+1}}) \in M_{i_1} \times \cdots \times M_{i_{k-s+1}}, i_t \in K\}$;

For $t = 1,...,s-1$:

$W_t = \{\{z_1,...,z_n\} : z_i \in W_{t-1}, i = 1,\ldots,n, \textit{ and } z_i \cap z_j = \emptyset, \forall i \neq j\}$;

$W_s = \{\{z_1,...,z_n\} : z_i \in W_{s-1}, i = 1,\ldots,n, \textit{ and } z_i \cap z_j = \emptyset, \forall\, i \neq j\ \}$;

Hence, W_0 is the set of all $(k-s+1)$-tuples and W_1 is the set of all subsets of W_0 that consists of n disjoint $(k-s+1)$-tuples. For $t = 2,...,s$, W_t is the set of all subsets of W_{t-1} that consist of exactly n disjoint members of W_{t-1}. Assume also a weight function $w : M_K \to \mathbb{R}$, where $M_K = \bigotimes_{i=1}^{k} M_i$.

Definition 7.2 *The $(k,s)\text{AP}_n$ problem asks for a minimum weight member of the set W_s.*

It follows that the $(k,s)AP_n$ problem asks for a minimum weight collection of n^s $(k-s+1)$-tuples. Note that an equivalent representation could be expressed in terms of asking for n^{s-1} clique partitions in the complete $(k-s+1)$-partite graph. The above definition also implies that the axial and planar problems ask for a minimum weight member of sets W_1 and W_2, respectively.

The Integer Programming formulation of the $(k,s)AP_n$ requires a binary variable $x_{m^1m^2\ldots m^k}$ and a weight coefficient $w_{m^1m^2\ldots m^k}$ for each k-tuple $(m^1, m^2,\ldots, m^k)$. Any $(k-s+1)$ out of the k n-sets $M_1,\ldots,M_k$ can be regarded as indexing the $(k-s+1)-$ tuples, while the remaining $(s-1)$ sets index the members of the powersets $W_1,\ldots,W_{s-1}$ selected. For example, $x_{m^1m^2\ldots m^k} = 1$ implies selection of the tuple $c = (m^1, m^2,\ldots, m^{k-s+1})$. Among the n (disjoint) members of set W_{s-1} selected, tuple c belongs to the m^kth one. This member of W_{s-1} consists of n disjoint members of W_{s-2} and tuple c belongs to the m^{k-1}th one, and so on.

We examine a simple corollary of the above. Let $c, d \in M_K$ and x_c, x_d be the corresponding variables. If $|c \cap d| \geq s$, the variables x_c, x_d have at least s indices in common, hence, assume $m^{i_1},\ldots,m^{i_s}$ to be these s common indices. The fact $x_c = x_d = 1$ implies the existence of a pair of $(k-s+1)$-tuples, both belonging to the same member m^{i_t} of powerset W_{i_t}, for all $t = 1,\ldots,s-1$. But then, member m^{i_1} of powerset W_1 contains two $(k-s+1)$-tuples having index m^{i_s} in common, i.e. two non-disjoint tuples, which is a contradiction to Definition 7.1. It follows that a pair of variables, which have at least s indices in common, cannot simultaneously take value 1. This fact is reflected in the integer programming formulation of $(k,s)AP_n$, which is derived by summing over all possible subsets of $k-s$ out of k indices.

$$\min \sum \{w_{m^1m^2\ldots m^k} \cdot x_{m^1m^2\ldots m^k} : (m^1, m^2, \ldots, m^k) \in M_K\}$$

$$\begin{array}{rcl}
\sum\{x_{m^1m^2\ldots m^k} & : & m^1 \in M_1, ..., m^{k-s-1} \in M^{k-s-1}, m^{k-s} \in M_{k-s}\} = 1 \\
 & & \text{for all } m^{k-s+1} \in M_{k-s+1}, m^{k-s+2} \in M_{k-s+2}, ..., m^k \in M_k \\
\sum\{x_{m^1m^2\ldots m^k} & : & m^1 \in M_1, ..., m^{k-s-1} \in M^{k-s-1}, m^{k-s+1} \in M_{k-s+1}\} = 1 \\
 & & \text{for all } m^{k-s} \in M_{k-s}, m^{k-s+2} \in M_{k-s+2}, ..., m^k \in M_k \\
 & & \vdots \\
\sum\{x_{m^1m^2\ldots m^k} & : & m^s \in M_s, m^{s+2} \in M_{s+2}, ..., m^k \in M_k\} = 1 \\
 & & \text{for all } m^1 \in M_1, ..., m^{s-1} \in M_s, m^{s+1} \in M_{s+1} \\
\sum\{x_{m^1m^2\ldots m^k} & : & m^{s+1} \in M_{s+1}, m^{s+2} \in M_{s+2}, ..., m^k \in M_k\} = 1 \\
 & & \text{for all } m^1 \in M_1, ..., m^{s-1} \in M_{s-1}, m^s \in M_s
\end{array} \tag{7.1}$$

$$x_{m^1m^2\ldots m^k} \in \{0,1\}, (m^1, m^2, \ldots, m^k) \in M_K\} \tag{7.2}$$

For any $S \subseteq K$, define $M_S = \bigotimes_{i \in S} M_i$. For $s \in K$, define $Q_{k,s} = \{S \subseteq K : |S| = s\}$, i.e. $Q_{k,s}$ is the set of all subsets of K with cardinality s, also called s-subsets. Observe that each constraint set corresponds to a single $S \in Q_{k,s}$. There are exactly s "fixed" indices in each constraint. The set of indices appearing in the sum is $M_{K \setminus S}$, whereas M_S is the set of indices common to all variables in a constraint. Let $A_n^{(k,s)}$ denote the $(0,1)$ matrix of the constraints (7.1). The matrix $A_n^{(k,s)}$ has n^k columns and $\binom{k}{s} \cdot n^s$ rows, i.e. n^s constraints for each of the $\binom{k}{s}$ distinct $S \in Q_{k,s}$. Each constraint involves n^{k-s} variables with coefficient 1.

Under these definitions, it is obvious that $(2,1)AP_n$ refers to the 2-index assignment problem, $(3,1)AP_n$ to the 3-index *axial* assignment problem ([9, 38]), $(3,2)AP_n$ to the 3-index *planar* assignment problem ([37, 63]), and $(4,2)AP_n$ to the 4-index *planar* assignment problem ([2]). Note that parameter s is central to the type of assignment required at each problem, i.e. the *axial* problems imply $s = 1$ and the *planar* problems imply $s = 2$. An alternative formulation and classification appears in [79].

7.2 Assignment polytopes and related structures

7.2.1 General concepts

Definitions from polyhedral theory were given in Section 3.1. Recall that, if $B^=$ denotes the matrix of the minimum equality system for a polyhedron $\mathbb{P}$, it holds that:

$$dim\mathbb{P} = n - rankB^= \tag{7.3}$$

The convex hull of the integer points satisfying the constraints (7.1) is the (k,s) *assignment polytope*, denoted as $P_I^{(k,s)}$. Formally, $P_I^{(k,s)} = conv\{x \in \{0,1\}^{n^k} : A_n^{(k,s)}x = e\}$, where e is a column vector of ones. The *linear relaxation* of $P_I^{(k,s)}$, also called the *linear assignment polytope*, is the polytope $P^{(k,s)} = \{x \in \mathbb{R}^{n^k} : A_n^{(k,s)}x = e, x \geq 0\}$. Obviously, $P_I^{(k,s)} \subseteq P^{(k,s)}$.

Clearly, the *assignment polytope* is a special case of the *set-partitioning* polytope defined as $P_{SP} = \{y \in \{0,1\}^q : By = e\}$, where B is a $0-1$ matrix. A close relative of P_{SP} is the *set-packing* polytope $\tilde{P}_{SP}$, defined exactly as P_{SP} but with "=" replaced by "$\leq$". In our case, $\tilde{P}_I^{(k,s)} = conv\{x \in \{0,1\}^{n^k} : A_n^{(k,s)}x \leq e\}$. A relation, inherited from the general case, is that $P_I^{(k,s)}$ is a face of $\tilde{P}_I^{(k,s)}$ implying $dimP_I^{(k,s)} \leq dim\tilde{P}_I^{(k,s)}$. Polytope $\tilde{P}_I^{(k,s)}$ is full dimensional, hence $dim\tilde{P}_I^{(k,s)} = n^k$, i.e. its dimension is independent of s. For a survey on $\tilde{P}_{SP}$, P_{SP} and on related problems, see [8].

Another class of polytopes related specifically to $P_I^{(k,1)}$ is given by the *multi-index generalised assignment* problems. The simplest representative is the 2-index generalised assignment problem *(GAP)* ([64]). These problems are defined for $s = 1$, i.e. they constitute an extension of the axial assignment problems. One of the constraint sets consists of equalities of the type " $= 1$", whereas the rest consist of inequalities of the type " $\leq b_t$", where $b_t \in \mathbb{Z}_+^{|M_t|}$, $t \in K \setminus \{1\}$. The IP model for the k-index generalised assignment problem is illustrated in [3]. Applications of this model, for $k = 3$, can be found in [41, 42, 64]. Finally, we note that a similar approach can be used for modelling an extension of the transportation problem, called the *solid (multi-index) transportation problem*, introduced in [48].

7.2.2 Two special cases

For $s = k$, constraints (7.1) reduce to the system of trivial equalities

$$x_{m^1m^2\dots m^k} = 1, \forall (m^1, m^2, \dots, m^k) \in M_K \tag{7.4}$$

whereas, for $s = 0$, constraints (7.1) result in the single equality constraint

$$\sum \{x_{m^1m^2\dots m^k} : (m^1, m^2, \dots, m^k) \in M_K\} = 1 \tag{7.5}$$

Lemma 7.1 *$rankA_n^{(k,k)} = n^k$ and $dimP^{(k,k)} = dimP_I^{(k,k)} = 0$.*

Proof. Trivially, $A_n^{(k,k)} = I(n^k)$. Therefore, $rankA_n^{(k,k)} = n^k$ and both polytopes contain the single point $x = e$. ■

Lemma 7.2 *$rankA_n^{(k,0)} = 1$ and $dimP^{(k,0)} = dimP_I^{(k,0)} = n^k - 1$.*

Proof. From (7.5), $rankA_n^{(k,0)} = 1$. The n^k unit vectors belong to both $P^{(k,0)}$ and $P_I^{(k,0)}$. The result follows from (7.3). ■

Corollary 7.3 *$P_I^{(k,0)}$ is a facet of $\tilde{P}_I^{(k,0)}$.*

Proof. $P_I^{(k,0)} \subset \tilde{P}_I^{(k,0)}$, since point $x = (0, \dots, 0)^T$ belongs to $\tilde{P}_I^{(k,0)}$ but not to $P_I^{(k,0)}$. The result follows from Lemma (7.2) and from the fact that $dim\tilde{P}_I^{(k,0)} = n^k$. ■

7.3 The (k,s) linear assignment polytope

This section establishes the dimension of polytope $P^{(k,s)}$ by determining the rank of matrix $A_n^{(k,s)}$. Before illustrating the main proof, certain intermediate results are presented.

First, we propose an ordering of the rows and columns of $A_n^{(k,s)}$. Observe that each set $S \in Q_{k,s}$ is uniquely associated to a row set of $A_n^{(k,s)}$. Each such S can be regarded as a subset of s indices, written (by convention) in ascending order $(i_1, \dots, i_s)$, where $i_t < i_{t+1}$ for $t = 1, \dots, s-1$.

Definition 7.3 *Let $S, S' \in Q_{k,s}$, where $S = (i_1, \dots, i_s)$, $S' = (i'_1, \dots, i'_s)$. It holds that $S > S'$ if and only if there exists $t \in \{1, \dots, s\}$ such that $i_p = i'_p$ for $p = 1, \dots, t-1$ and $i_t > i'_t$.*

It follows that there exists a strict lexicographic order on all subsets of s indices, i.e. a strict order of all $S \in Q_{k,s}$. Based on this ordering, row sets of $A_n^{(k,s)}$ are positioned in descending order. Within a particular row set, a row corresponds to an s-tuple of the set M_S. These rows are placed in ascending order with respect to the corresponding s-tuples. Columns of $A_n^{(k,s)}$ appear also in lexicographic order.

Example 7.1 *For $k = 4, s = 2$, there exist $\binom{4}{2} = 6$ row sets. They are considered in descending order with respect to the pair of "fixed" indices, i.e. $\{(m^3, m^4), (m^2, m^4), (m^2, m^3), (m^1, m^4), (m^1, m^3), (m^1, m^2)\}$. Rows in, say, the second row set are identified by the values of the pair of indices (m^2, m^4). The values of this pair of indices are considered in ascending order, i.e. $(1,1), \dots, (1,n), \dots, (n,1), \dots, (n,n)$. The columns of $A_n^{(4,2)}$ are ordered lexicographically with respect to the values of the 4-tuple (m^1, m^2, m^3, m^4), i.e. $(1,1,1,1)$, $\dots$, $(1,1,1,n)$, $\dots$, $(n,1,1,1)$, $\dots, (n,n,n,n)$.*

Next, we wish to determine the exact position of each row set in the matrix $A_n^{(k,s)}$. Let $r_0 = (i_1, \dots, i_s) \in Q_{k,s}$, where $i_1 < i_2 < \dots < i_s$. Define the function $f : Q_{k,s} \to \{1, \dots, |Q_{k,s}|\}$, where $|Q_{k,s}| = \binom{k}{s}$:

$$f(i_1, \dots, i_s) = \sum_{t=1}^{s} \left\{ \sum_{j=i_t+1}^{k-(s-t)} \binom{k-j}{s-t} \right\} + 1, \quad i_1 < \dots < i_s,\ s, k \in \mathbb{Z}_+ \tag{7.6}$$

We show that f provides the position of row set r_0 in matrix $A_n^{(k,s)}$, i.e. it is equal to the number of row sets preceding r_0 plus one. Let r_1 be a preceding row set indexed by the same indices as r_0 in positions $1, \dots, t$ and indices $[k-(s-t)+1], \dots, k$ in positions $t+1, \dots, s$, i.e. $r_1 = (i_1, i_2, \dots, i_t, k-(s-t)+1, \dots, k)$. Any row set preceding r_1, which has the same indices as r_0 in positions $1, \dots, t-1$, must have an index j in position t, such that $j \in \{i_t+1, \dots, k-(s-t)\}$. For each such j, positions $t+1, \dots, s$, at the s-tuple indexing the row set, can be completed by choosing $s-t$ out of $k-j$ indices. This holds because there are $s-t$ positions to be filled and $k-j$ indices with value larger than j. This results in $\binom{k-j}{s-t}$ possible row sets for each $j \in \{i_t+1, \dots, k-(s-t)\}$. Repeating this process for all $i_t, t = 1, \dots, s$, and taking the sum for all i_t, results in the total number of row sets preceding r_0. Therefore, the row set $(i_1, \dots, i_s)$, $i_1 < \dots < i_s$, includes rows $f(i_1, \dots, i_s) \cdot n^s + 1$ to $(f(i_1, \dots, i_s) + 1) \cdot n^s$, i.e. one row for each value of the s-tuple $(m^{i_1}, \dots, m^{i_s})$. It can be seen that, regarding the first and the last row set, $f(k-s+1, \dots, k) = 1$ and $f(1, \dots, s) = \binom{k}{s}$ respectively, as required.

The following lemma establishes an equality, which is to be utilised later in this section.

Lemma 7.4 *Let $k \in Z_+ \backslash \{0, 1\}$. For integers s, r, where $r \leq s \leq k-1$, the following identity holds*

$$(-1)^{s-r} \cdot \sum_{l=s-r}^{s} \binom{k-(s-l+1)}{l} \binom{l}{s-r} = \sum_{l=r}^{s} \binom{k}{l} \cdot \binom{l}{l-r} \cdot (-1)^{l-r} \tag{7.7}$$

Proof. Equation (7.7) can be re-written as:

$$\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l}\binom{l}{s-r}=\sum_{l=r}^{s}\binom{k}{l}\cdot\binom{l}{r}\cdot(-1)^{s-l} \tag{7.8}$$

We proceed by induction on parameter s. For $s=0$, both sides of eq. (7.8) are equal to 1. Assuming that (7.8) holds for certain s, we prove that it is also true for $s+1$, i.e.:

$$\sum_{l=s+1-r}^{s+1}\binom{k-(s-l+2)}{l}\binom{l}{s+1-r}=\sum_{l=r}^{s+1}\binom{k}{l}\cdot\binom{l}{r}\cdot(-1)^{s+1-l} \tag{7.9}$$

The right-hand side of (7.9) can be written as:

$$\begin{aligned}
&\sum_{l=r}^{s+1}\binom{k}{l}\cdot\binom{l}{r}\cdot(-1)^{s+1-l}\\
=\;&(-1)\cdot\sum_{l=r}^{s+1}\binom{k}{l}\cdot\binom{l}{r}\cdot(-1)^{s-l}\\
=\;&(-1)\sum_{l=r}^{s}\binom{k}{l}\cdot\binom{l}{r}\cdot(-1)^{s-l}+\binom{k}{s+1}\cdot\binom{s+1}{r}
\end{aligned} \tag{7.10}$$

Since (7.8) holds for s, the first term of (7.10) can be replaced by its equal. Hence, we obtain that:

$$\sum_{l=r}^{s+1}\binom{k}{l}\cdot\binom{l}{r}\cdot(-1)^{s+1-l}=(-1)\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l}\binom{l}{s-r}+\binom{k}{s+1}\cdot\binom{s+1}{r} \tag{7.11}$$

Given (7.11), (7.9) can be written equivalently as:

$$\sum_{l=s+1-r}^{s+1}\binom{k-(s-l+2)}{l}\binom{l}{s+1-r}+\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l}\binom{l}{s-r}=\binom{k}{s+1}\cdot\binom{s+1}{r}$$

or

$$\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l+1}\binom{l+1}{s+1-r}+\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l}\binom{l}{s-r}=\binom{k}{s+1}\cdot\binom{s+1}{r}$$

By replacing l by $s-l$ in this last expression and including both terms in a single summand, we obtain:

$$\sum_{l=0}^{r}\left\{\binom{k-1-l}{s+1-l}\binom{s+1-l}{s+1-r}+\binom{k-1-l}{s-l}\binom{s-l}{s-r}\right\}=\binom{k}{s+1}\cdot\binom{s+1}{r} \tag{7.12}$$

Observe that $\binom{k-1-l}{s-l}=\binom{k-1-l}{s+1-l}\cdot\frac{s+1-l}{k-s-1}$ and $\binom{s-l}{s-r}=\binom{s+1-l}{s+1-r}\cdot\frac{s+1-r}{s+1-l}$. Hence, equation (7.12) is equivalent to

$$\sum_{l=0}^{r}\left\{\binom{k-1-l}{s+1-l}\binom{s+1-l}{s+1-r}\left(1+\frac{s+1-r}{k-s-1}\right)\right\}=\binom{k}{s+1}\cdot\binom{s+1}{r}$$

Term $\left(1+\frac{s+1-r}{k-s-1}\right)$ is independent of l, thus can be placed outside of the sum. After performing certain arithmetic operations, we obtain the following equation.

$$\sum_{l=0}^{r}\left\{\binom{k-1-l}{s+1-l}\cdot\binom{s+1-l}{s+1-r}\right\}=\frac{k-s-1}{k-r}\cdot\binom{k}{s+1}\cdot\binom{s+1}{r} \tag{7.13}$$

To summarise the steps up to this point, we have to illustrate the inductive step of our proof, which is formalised by equation (7.9). After various transformations, we have shown that (7.9) holds if and only if (7.13) holds. To prove the validity of (7.13), it is sufficient to examine the values of $r = 0, ..., s$ by induction. The case of $r = 0$ is trivial. Assuming that (7.13) holds for a certain r, we can prove its validity for $r + 1$ by applying a transformation analogous to the one applied to derive equation (7.12) from equation (7.9). This shows the validity of the induction on parameter s. The proof of (7.7) is now complete. ∎

Theorem 7.5 $rankA_n^{(k,s)} = \sum_{r=0}^{s}\{\binom{k}{r}(n-1)^r\}$.

Proof. For $s = k$ and $s = 0$ the theorem stands by virtue of Lemmas 7.1 and 7.2, respectively. For $1 \le s \le k-1$, the proof includes $s+1$ steps. Let $r = 0, \ldots, s$ denote the step counter. At each step we remove sets of linearly dependent rows. The number of remaining rows provides an upper bound on $rankA_n^{(k,s)}$. Next, we identify a non-singular square submatrix consisting of these rows and an equal number of columns. This approach was followed in the proof of Theorem 3.2, which is actually a special case of this theorem.

Observe first that the sum of all equalities belonging to the same row set yields an equality stating that the sum of all variables is equal to n^s. Therefore, we remove the first row from all the row sets except for the first one. The total number of rows removed is $\binom{k}{s} - 1 = \binom{k}{0}(n-1)^0(\binom{k}{s} - 1)$. This is the step $r = 0$.

At step $r = 1$, for each index $q \in K$, consider the subset of row sets $R_q = \{S \in Q_{k,s} : q \in S\}$, $|R_q| = \binom{k-1}{s-1}$. Each row set belonging to R_q is further partitioned into n subsets, one for each value of the index m^q. The rows of the row sets, belonging to the same subset, form a $(k-1, s-1)AP_n$: index m^q is fixed, therefore its presence in both sets S and K can be ignored. Observe that the n $(k-1, s-1)AP_n$ problems formed in this way, (i.e. one problem for each value of m^q), are independent of each other, since each problem has a distinct set of variables and constraints. Each such problem consists of $\binom{k-1}{s-1}$ row sets, the sum of rows in each row set stating that the sum of all the n^{k-1} variables is equal to n^{s-1}. Hence, for each such problem, except for the $(k-1, s-1)AP_n$ defined for $m^q = 1$, we remove the first row from each row set, excluding the first row set. Note that the "first" row set is the one appearing first in matrix $A_n^{(k,s)}$, according to the ordering described previously. In other words, for each of the $n-1$ $(k-1, s-1)AP_n$ problems, we remove $\binom{k-1}{s-1} - 1$ rows. Hence, for each $q \in K$, we remove $(n-1)(\binom{k-1}{s-1} - 1)$ rows, yielding a total of $k(n-1)(\binom{k-1}{s-1} - 1) = \binom{k}{1}(n-1)^1(\binom{k-1}{s-1} - 1)$ rows removed in this step.

Example 7.1 (cont.) *The row sets* $\{(m^1, m^4), (m^1, m^3), (m^1, m^2)\}$ *form* n *independent* $(3,1)\text{AP}_n$, *one for each value of index* m^1. *For all* $m^1 \neq 1$, *a row is removed from each of the sets* $\{(m^1, m^3), (m^1, m^2)\}$, *i.e. a total of* $2(n-1)$ *rows.*

Similarly, at step r ($r \le s$), sets of exactly r distinct indices are considered. Let $(i_1, \ldots, i_r)$ denote such a subset. We define the subset of row sets $R_{i_1,\ldots,i_r} = \{S \in Q_{k,s} : \bigcup_{p=1}^{r}\{i_p\} \subseteq S\}$, $|R_{i_1,\ldots,i_r}| = \binom{k-r}{s-r}$. Each row set belonging to $R_{i_1,\ldots,i_r}$ is further partitioned into n^r subsets, one for each value of the r-tuple $(m^{i_1}, \ldots, m^{i_r}) \in M_{i_1} \times \cdots \times M_{i_r}$. Again, the rows of the row sets, belonging to the same subset (identified by the same value of the r-tuple $(m^{i_1}, \ldots, m^{i_r})$) form a $(k-r, s-r)AP_n$. Since there are n^r distinct subsets, we have n^r independent $(k-r, s-r)AP_n$. Each $(k-r, s-r)AP_n$ consists of $\binom{k-r}{s-r}$ row sets, the sum of rows in each row set stating that the sum of all the n^{k-r} variables is equal to n^{s-r}. Hence, for each of these problems, excluding the ones defined by the r-tuples $(m^{i_1}, \ldots, m^{i_r})$ for which *at least one* of the indices is equal to 1, we remove the first row from each of its last $\binom{k-r}{s-r} - 1$ row sets. Therefore, for each $(k-r, s-r)AP_n$, defined by the r-tuple $(m^{i_1}, \ldots, m^{i_r})$, $m^{i_p} \neq 1$, $\forall p = 1, \ldots, r$, we remove $\binom{k-r}{s-r} - 1$ rows. The exact rows removed for the problem identified by $(m^{i_1}, \ldots, m^{i_r})$ are defined via the following remark.

Remark 7.1 *Let t^* denote the maximal element of $R_{i_1,\ldots,i_r}$, i.e. $t^* \geq t$ for all $t \in R_{i_1,\ldots,i_r}$ (Definition 7.3). For each $t \in R_{i_1,\ldots,i_r} \backslash \{t^*\}$, we remove the row which has $m^j = 1$ for every $j \in t \backslash \{i_1, \ldots, i_r\}$ and $(m^{i_1}, \ldots, m^{i_r})$ in positions $(i_1, \ldots, i_r)$, where $m^{i_p} \neq 1$, $\forall p = 1, \ldots, r$. There is a $1-1$ correspondence between each $t \in R_{i_1,\ldots,i_r} \setminus \{t^*\}$ and each row removed. Obviously, $|R_{i_1,\ldots,i_r} \setminus \{t^*\}| = \binom{k-r}{s-r} - 1$.*

Because there are $(n-1)^r$ such problems, we remove $(n-1)^r \cdot (\binom{k-r}{s-r} - 1)$ rows for each distinct r-set $(i_1, \ldots, i_r)$. There are $\binom{k}{r}$ distinct r-sets of indices, therefore the number of rows removed at step r is at most

$$\binom{k}{r} \cdot (n-1)^r \cdot (\binom{k-r}{s-r} - 1) \tag{7.14}$$

Assume, without loss of generality, that row removal is implemented (i) in increasing order with respect to r and (ii) for a certain r, in increasing order with respect to row ordering. For example, the step $r = 1$ is preceding step $r = 2$ and the $(k-2, s-2)AP_n$ problems corresponding to the pair of indices (m^1, m^2) are considered before the $(k-2, s-2)AP_n$ problems corresponding to the pair (m^1, m^3). To show that the upper bound defined by (7.14) is attainable, we need to prove the following lemma.

Lemma 7.6

(i) *No rows of the $(k-r, s-r)\mathrm{AP}_n$ problems indexed by the r-tuple r_0 were removed at a previous step involving an r-tuple $\hat{r}_0$ examined before r_0.*

(ii) *No rows of the $(k-r, s-r)\mathrm{AP}_n$ problems indexed by the r-tuple r_0 were removed at a previous step involving a v-tuple v_0 where $v < r$.*

Proof. To prove (i), let $r_0 = (i_1, \ldots, i_r)$, $\hat{r}_0 = (\hat{i}_1, \ldots, \hat{i}_r)$ be two distinct r-tuples, i.e. there exists at least one $q \in \{1, \ldots, r\}$, such that $i_q \neq \hat{i}_q$. Let $\hat{r}_0$ be considered first and let t be an s-tuple, indexing a row set, such that $r_0, \hat{r}_0 \subseteq t$. If $t = t^*$ (or $t = \hat{t}^*$), where t^* $(\hat{t}^*)$ is the maximal element of $R_{i_1,\ldots,i_r}$ $(R_{\hat{i}_1,\ldots,\hat{i}_r}$ respectively), then no row is removed from row set t with respect to r_0 $(\hat{r}_0)$. Thus, we assume that $t \neq t^*, \hat{t}^*$. This implies that the rows of t belong to the $(k-r, s-r)AP_n$ problems considered for both r_0 and $\hat{r}_0$. Let $m^{\hat{r}_0} = (m^{\hat{i}_1}, \ldots, m^{\hat{i}_r})$ and $m^{r_0} = (m^{i_1}, \ldots, m^{i_r})$. For each value of $m^{\hat{r}_0}$, where $m^{\hat{i}_p} \neq 1$, for all $p = 1, \ldots, r$, a row having $m^j = 1$, for all $j \in t \setminus \hat{r}_0$ is removed (Remark 7.1). This implies that this row has $m^j = 1$ for all $j \in r_0 \setminus (r_0 \cap \hat{r}_0)$ Recall that when considering r_0, only $(k-r, s-r)AP_n$ problems corresponding to values of m^{r_0} with no index equal to 1 are examined. Therefore, all rows removed when considering r-tuples $m^{\hat{r}_0}$ belong to $(k-r, s-r)AP_n$ problems not examined when considering r-tuples m^{r_0}.

The proof of (ii) follows the same idea. Assume $r_0 = (i_1, \ldots, i_r)$ and $v_0 = (\hat{i}_1, \ldots, \hat{i}_v)$, $v < r$. There exists at least one index i_t belonging to r_0 but not to v_0. Let also t be an s-tuple indexing a row set such that $r_0, v_0 \subset t$. If a row is deleted when considering the $(k-v, s-v)AP_n$ corresponding to a certain value m_{v_0}, not having any index equal to 1, then all indices of $t \setminus v_0$ will have value 1 (Remark 7.1). This includes the value of the index m^{i_t}, $i_t \in r_0 \setminus v_0$. But then this row would belong to the problem $(k-r, s-r)AP_n$ corresponding to a value of m^{r_0}, where at least one of the indices has value 1. This problem is not examined when considering r_0, since all problems defined have $m^{i_t} \neq 1$. ■

(Back to the proof of Theorem 7.5) Summing over all r, we obtain a total number of linearly dependent rows of $A_n^{(k,s)}$ equal to

$$\sum_{r=0}^{s}\left\{\binom{k}{r}\cdot(n-1)^r\cdot(\binom{k-r}{s-r}-1)\right\}$$

which yields the following upper bound on the $rankA_n^{(k,s)}$:

$$\begin{aligned} rankA_n^{(k,s)} &\leq \binom{k}{s}\cdot n^s - \sum_{r=0}^{s}\left\{\binom{k}{r}\cdot(n-1)^r\cdot(\binom{k-r}{s-r}-1)\right\} \\ &= \binom{k}{s}\cdot\sum_{r=0}^{s}\binom{s}{r}\cdot(n-1)^r - \sum_{r=0}^{s}\binom{k}{r}\binom{k-r}{s-r}\cdot(n-1)^r \\ &\quad +\sum_{r=0}^{s}\binom{k}{r}\cdot(n-1)^r \end{aligned}$$

By using the property $\binom{k}{r}\binom{k-r}{s-r} = \binom{k}{s}\binom{s}{r}$, we obtain

$$\begin{aligned} rankA_n^{(k,s)} &\leq \sum_{r=0}^{s}\binom{k}{s}\binom{s}{r}\cdot(n-1)^r - \sum_{r=0}^{s}\binom{k}{s}\binom{s}{r}\cdot(n-1)^r \\ &\quad +\sum_{r=0}^{s}\binom{k}{r}\cdot(n-1)^r \end{aligned}$$

or

$$rankA_n^{(k,s)} \leq \sum_{r=0}^{s}\binom{k}{r}\cdot(n-1)^r \tag{7.15}$$

To complete the proof, we must provide an equal lower bound on $rankA_n^{(k,s)}$. For this purpose, we exhibit a number of linearly independent columns equal to the number of independent rows exhibited previously. Each column is presented using the notation $(m^1, \ldots, m^{k-s}, m^{k-s+1}, \ldots, m^k)$, i.e. it is important to differentiate between the values of the first $k-s$ indices and the values of the last s indices. Consider the following (disjoint) sets of columns.

a) Columns having the first $k-s$ indices equal to 1 and all possible values for the last s indices in increasing order. These columns are

$$(1, \ldots, 1, 1, \ldots, 1), \ldots, (1, \ldots, 1, n, \ldots, n)$$

There are n^s such columns.

b) Columns having any $k-s-1$ of the first $k-s$ indices equal to 1 and all possible values, except for 1, for the remaining index. These columns also have $m_{k-s+1} = 1$ and all possible values, including 1, for the last $s-1$ indices. An example is the subset of columns

$$\begin{array}{rcl} \text{Columns} & : & (\underbrace{1,\ldots,1,2}, 1, 1\ldots,1),\ldots,(1,\ldots,1,2,1,n,\ldots,n) \\ \text{No. of indices} & : & k-s \end{array}$$

The cardinality of a given subset of columns which has the first $k-s$ indices fixed is n^{s-1}. There are $n-1$ such subsets that have the value 1 in the same $k-s-1$ out of the first $k-s$ indices. There are $k-s$ distinct configurations, for which the $k-s-1$ out of the first $k-s$ indices have value 1 and the remaining index has a value different than one. Thus, the total number of columns in this set is $(k-s)\cdot(n-1)\cdot n^{s-1}$.

c) Columns having any $k-s-1$ of the first $k-s-1+r$ $(2 \leq r \leq s-1)$ indices equal to 1 and any possible value, except for 1, for the r remaining indices $m^{i_1},\ldots,m^{i_r}, i_q \in \{1,\ldots,k-s-1+r\}$, for all $q \in \{1,\ldots,r\}$. These columns also have $m^{k-s+r}=1$ and any possible value for the last $s-r$ indices. For example, let $r=2$ and let the last two, of the first $k-s-1+2$, indices have values different from 1. Then the subset of columns defined is

$$\begin{array}{rcl} \text{Columns} & : & (\underbrace{1,\ldots,1,2,2}_{k-s-1+2}, 1, 1,\ldots,1),\ldots,(\underbrace{1,\ldots,1,n,n}_{k-s-1+2}, 1, n,\ldots,n) \\ \text{No. of indices} & : & \end{array}$$

There are $\binom{k-(s-r+1)}{r}$ options for selecting the indices $m^{i_1},\ldots,m^{i_r}$ and, for each such option, $n^{s-r}\cdot(n-1)^r$ columns are included.

d) Columns having any $k-s-1$ of the first $k-1$ indices equal to 1 and any possible value, except for 1, for the s remaining indices $m^{i_1},\ldots,m^{i_s}, i_q \in \{1,\ldots,k-1\}$, for all $q \in \{1,\ldots,s\}$. These columns also have $m^k=1$. There are $\binom{k-1}{s}$ options for selecting the indices $m^{i_1},\ldots,m^{i_s}$ and for each option $(n-1)^s$columns are included.

The total number of columns exhibited in (a),...,(d) is

$$\begin{aligned} &\sum_{r=0}^{s}\binom{k-(s-r+1)}{r}\cdot n^{s-r}\cdot(n-1)^r \\ = &\sum_{r=0}^{s}\left\{\binom{k-(s-r+1)}{r}\cdot n^{s-r}\cdot\sum_{l=0}^{r}\left\{\binom{r}{l}\cdot(-1)^l\cdot n^{r-l}\right\}\right\} \end{aligned} \tag{7.16}$$

The right-hand side of (7.16) can be written in the form $\sum_{r=0}^{s} n^r \cdot g(r)$. To identify $g(r)$, observe that n^i, for all $i \in \{0,\ldots,s\}$, can be derived as $n^i\cdot n^0$, for $r=s-i$ and $l=r=s-i$ (Table 7.2). The binomial coefficients of (7.16) will then be $\binom{k-(s-(s-i)+1)}{s-i}=\binom{k-i-1}{s-i}$ and $\binom{s-i}{s-i}\cdot(-1)^{s-i}$.

It follows that $g(r)=(-1)^{s-r}\cdot\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l}\binom{l}{s-r}$. Thus, (7.16) becomes

$$\begin{aligned} &\sum_{r=0}^{s}\binom{k-(s-r+1)}{r}\cdot n^{s-r}\cdot(n-1)^r \\ = &\sum_{r=0}^{s} n^r\cdot(-1)^{s-r}\cdot\sum_{l=s-r}^{s}\binom{k-(s-l+1)}{l}\binom{l}{s-r} \end{aligned} \tag{7.17}$$

n^i	m	l	${}^{k-(p-m+1)}C_m$	${}^{m}C_l$
$n^i \cdot n^0$	$p-i$	$p-i$	${}^{k-i-1}C_{p-i}$	${}^{p-i}C_{p-i} \cdot (-1)^{p-i}$
$n^{i-1} \cdot n^1$	$p-i+1$	$p-i$	${}^{k-i}C_{p-i+1}$	${}^{p-i+1}C_{p-i} \cdot (-1)^{p-i}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n^0 \cdot n^i$	p	$p-i$	${}^{k-1}C_p$	${}^{p}C_{p-i} \cdot (-1)^{p-i}$

Table 7.2:

Expressing term $(n-1)^r$ in (7.15) using Newton's polynomial, we obtain

$$\begin{aligned}\sum_{r=0}^{s}\binom{k}{r}\cdot(n-1)^r &= \sum_{r=0}^{s}\binom{k}{r}\cdot\left\{\sum_{l=0}^{r}\binom{r}{l}\cdot(-1)^{n-l}\cdot n^l\right\}\\ &= \sum_{r=0}^{s}n^r\cdot\sum_{l=r}^{s}\binom{k}{l}\cdot\binom{l}{l-r}\cdot(-1)^{l-r} \qquad (7.18)\end{aligned}$$

It follows from Lemma 7.4 that the coefficients of n^r in the right-hand sides of (7.17) and (7.18) are equal. Hence, we obtain the following identity:

$$\sum_{r=0}^{s}\binom{k}{r}\cdot(n-1)^r = \sum_{r=0}^{s}\binom{k-(s-r+1)}{r}\cdot n^{s-r}\cdot(n-1)^r$$

The submatrix of $A_n^{(k,s)}$ formed by these columns and the remaining rows is square upper triangular with each diagonal entry equal to 1, therefore non-singular. This provides a lower bound on $rankA_n^{(k,s)}$equal to the upper bound illustrated in (7.15). The proof is complete. ■

Corollary 7.7 $dimP^{(k,s)} = \sum_{r=0}^{k-s-1}\binom{k}{r}\cdot(n-1)^{k-r}$.

Proof. Follows from (7.3) by expressing term n^k using Newton's polynomial: $n^k = \sum_{r=0}^{k}\binom{k}{r}\cdot(n-1)^k$. ■

7.4 The (k,s) assignment polytope

Amongst assignment problems, the one with the longest history is the two-index assignment problem. From early on, it was known that $P_I^{(2,1)} \equiv P^{(2,1)}$. This is a direct consequence of Birkhoff's theorem on *permutation* matrices (see [22, Theorem 1.1]). Hence, $dimP_I^{(2,1)} = dimP^{(2,1)}$, which by Corollary 7.7 is equal to $(n-1)^2$. In Table 7.3, we state the known values of $dimP_I^{(k,s)}$ for given (k,s), other than $(2,1)$. We also state the values of n for which the proofs are valid, along with the appropriate references.

Table 7.3: Known values of $dimP_I^{(k,s)}$.

(k,s)	n	$dimP_I^{(k,s)}$	References
$(k,0), \forall k \in Z_+$	≥ 0	n^k-1	Lemma 7.2
$(3,1)$	≥ 3	n^3-3n+2	R. Euler ([38]), E. Balas et al. ([9])
$(3,2)$	≥ 4	$(n-1)^3$	R. Euler et al.([37])
$(4,2)$	$\geq 4, \neq 6$	n^4-6n^2+8n-3	G. Appa et al.([2])
$(k,k), \forall k \in Z_+$	≥ 0	0	Lemma 7.1

For general (k,s), Corollary 7.7 implies $dimP_I^{(k,s)} \leq \sum_{r=0}^{k-s-1} \binom{k}{r} \cdot (n-1)^{k-r}$, since $P_I^{(k,s)} \subseteq P^{(k,s)}$. However, establishing the exact value of $dimP_I^{(k,s)}$ is not straightforward. Note also that $P_I^{(k,s)} = \emptyset$ for certain values of k, s, n (for example, consider the polytope for $k=4, s=2, n=2$).

First, let us take a closer look at the integer vectors of $P_I^{(k,s)}$. Definition 7.1 implies that n^s k-tuples have to be selected, therefore an integer vector must have exactly n^s variables equal to 1. Moreover, any integer vector must signify n disjoint members of powerset W_{s-1}, each one containing n^{s-1} tuples (i.e. variables). Since any index $m^t \in M_t$, $t \in K$, can be regarded as indexing powerset W_{s-1}, any of its values m_0^t must appear in exactly n^{s-1} non-zero variables. Hence, at an integer point of $P_I^{(k,s)}$, each element of any of the sets $M_1, \ldots, M_k$ appears in exactly n^{s-1} variables set to one. This remark is very useful in terms of providing a mechanism for transition from one integer point to another. Consider an arbitrary integer point $x \in P_I^{(k,s)}$, and a pair of index values $m_0^t, m_1^t \in M_t, t \in K$. We set $m^t = m_1^t$ for all the $(k-s+1)$-tuples with $m^t = m_0^t$ and also $m^t = m_0^t$ for all the $(k-s+1)$-tuples with $m^t = m_1^t$. The derived structure corresponds to another integer point $\hat{x} \in P_I^{(k,s)}$, since there are again exactly n^{s-1} "new" variables set to one with $m^t = m_0^t$ and n^{s-1} "new" variables set to one with $m^t = m_1^t$. The variables, with $m^t = m_0^t$ and $m^t = m_1^t$, set to one at point x, are set to zero at point $\hat{x}$ and vice versa. The values for the rest of the variables remain the same at both points. This notion of interchanging the role of two index values is formalised with the introduction of the *interchange* operator ($\leftrightarrow$). Hence, by setting $\hat{x} = x(m_0^t \leftrightarrow m_1^t)_t$, we imply that, at point x, we interchange the index values m_0^t and m_1^t, thus deriving point $\hat{x}$. The *conditional interchange* is defined accordingly (See Section 3.2).

Given (k,s,n), an important issue is whether $P_I^{(k,s)} \neq \emptyset$. This question is difficult to answer for general (k,s,n). However, for certain values of s, we know that $P_I^{(k,s)} \neq \emptyset$. We have seen two such cases in Section 7.2.2. Another such case is obtained for $s=1$. It is easy to see that $P_I^{(k,1)} \neq \emptyset$, $\forall k, n \in \mathbb{Z}_+$. This is because the *diagonal* solution, noted as $\hat{x}$ $(x_{11\cdots1} = \cdots = x_{nn\cdots n} = 1)$, always satisfies constraints (7.1), (7.2), for $s=1$. For general $s \geq 2$, we provide the following necessary condition for $P_I^{(k,s)}$ to be non-empty.

Proposition 7.8 *For $s \geq 2$, a necessary condition for the existence of a solution to the $(k,s)\mathrm{AP}_n$ is $k \leq n+s-1$.*

Proof. Observe first that, according to (7.1), variables appear at the same row if and only if they have at least s indices in common (common values for indices belonging to m^S).

Given that $P_I^{(k,s)} \neq \emptyset$ and $s \geq 2$, consider the point $x' \in P_I^{(k,s)}$ having $x_{1\cdots1m_1^s\cdots m_1^k} = x_{1\cdots1m_2^s\cdots m_2^k}$ $= \cdots = x_{1\cdots1m_n^s\cdots m_n^k} = 1$. Such a point always exists, since rows $(1,\ldots,1,m_j^s) \in M_1 \times M_2 \cdots \times M_s$, $j=1,\ldots,n$ must have exactly one variable equal to 1. Also note that all indices $m_j^t, j=1,\ldots,n$, must be pairwise different since otherwise a constraint would have a left-hand side equal to 2. Consider the following sequence of points:

$$\begin{aligned}
x^s &= x'(m_1^s \neq 1?m_1^s \leftrightarrow 1)_s(m_2^s \neq 2?m_2^s \leftrightarrow 2)_s \cdots (m_n^s \neq n?m_n^s \leftrightarrow n)_s \\
x^{s+1} &= x^s(m_1^{s+1} \neq 1?m_1^{s+1} \leftrightarrow 1)_{s+1}(m_2^{s+1} \neq 2?m_2^{s+1} \leftrightarrow 2)_{s+1} \cdots (m_n^{s+1} \neq n?m_n^{s+1} \leftrightarrow n)_{s+1} \\
&\vdots \\
x^k &= x^{k-1}(m_1^k \neq 1?m_1^k \leftrightarrow 1)_k(m_2^k \neq 2?m_2^k \leftrightarrow 2)_k \cdots (m_n^k \neq n?m_n^k \leftrightarrow n)_k
\end{aligned}$$

Let $x = x^k$. At point x we have $x_{1\cdots11\cdots1} = x_{1\cdots12\cdots2} = \cdots = x_{1\cdots1n\cdots n} = 1$. The notation implies that the first $s-1$ indices are equal to 1 and the indices $s,\ldots,k$ have the same value. Point x must satisfy all constraints including row $(2, 1, \dots, 1) \in M_1 \times M_2 \cdots \times M_s$. Consequently, there must be exactly one variable of the form $x_{21\dots1m^{s+1}\dots m^k}$ with value 1. None of the indices $m^{s+1}, \dots, m^s$ can take the value 1, because the left-hand side of a constraint would then become 2. For the same reason they must be pairwise different. Therefore, at most $n-1$ values must be allocated to $k-s$ indices. This implies $k - s \leq n - 1 \Leftrightarrow k \leq s + n - 1$. ■

For $s = 2$, this theorem states the well-known fact that there can exist no more than $n-1$ *MOLS* of order n ([60, Theorem 2.1]).

Let us explore the relationship between the $(k, s)AP_n$ and the problems arising when decreasing by one any of the parameters k, s. First recall that a member of powerset W_s is equivalent to n disjoint members of powerset W_{s-1}, i.e. to n sets of n^{s-1} $(k - s + 1)$ tuples each. It follows that a solution of $(k, s)AP_n$ is equivalent to n disjoint solutions of $(k-1, s-1)AP_n$. Equivalently, a vector $x \in P_I^{(k,s)}$ can be used to construct n linearly independent vectors $\bar{x}^i \in P_I^{(k-1,s-1)}$, $i = 1, \dots, n$. The construction is quite straightforward: for a certain index $m^t, t \in K$, and $x \in P_I^{(k,s)}$, define

$$\bar{x}^i_{m^1\dots m^{t-1}m^{t+1}\dots m^k} = x_{m^1\dots m^{t-1}m^t_i m^{t+1}\dots m^k}$$

for all $(m^1, \dots, m^{t-1}, m^t, \dots, m^k) \in M_{K\setminus\{t\}}$ and $i = 1, \dots, n$. It is easy to verify that $\bar{x}^i \in P_I^{(k-1,s-1)}$, for $i = 1, \dots, n$. As an example, observe that a solution to $(4, 2)AP_n$ (OLS) is equivalent to n independent solutions of $(3, 1)AP_n$ (i.e. n disjoint transversals). Let $x \in P_I^{(4,2)}$ and $m^1, \dots, m^4$ be the four indices. For $m^4 = m^4_i, i = 1, \dots, n$, define the vector $\bar{x}^i_{m^1m^2m^3} = x_{m^1m^2m^3m^4_i}$. It is not difficult to see that $\bar{x}^i \in P_I^{(3,1)}$, $i = 1, \dots, n$.

In an analogous way, a solution of $(k, s)AP_n$ implies a solution to $(k-1, s)AP_n$. As an example, let $s = 2$ and notice that a solution to $(4, 2)AP_n$ implies a solution to $(3, 2)AP_n$, i.e. a pair of OLS implies a certain Latin square. Observe that the $(k-1, s)AP_n$ requires an integer solution with n^s variables set to one, but with one index less (i.e. n^s $(k-s)$ tuples). Let the index dropped be $m^t \in M_t$, $t \in K$. Assuming $x \in P_I^{(k,s)}$, a (single) vector $x' \in P_I^{(k-1,s)}$ is derived as follows:

$$x'_{m^1\dots m^{t-1}m^{t+1}\dots m^k} = \left\{ \begin{array}{ll} 1, & \text{if } \exists\, m^t_0 \in M_t : \ x_{m^1\dots m^{t-1}m^t_i m^{t+1}\dots m^k} = 1 \\ 0, & \text{otherwise} \end{array} \right\},$$

for all $(m^1, \dots, m^{t-1}, m^t, \dots, m^k) \in M_{K\setminus\{t\}}$

In contrast, a solution of $(k, s)AP_n$ cannot give rise to a solution of $(k, s-1)AP_n$ and vice versa. Note also that the inverse of the above is not true, i.e. a single solution to either $(k-1, s-1)AP_n$ or $(k-1, s)AP_n$ is not always extendable to a solution of the $(k, s)AP_n$. The hierarchy implied by these observations is illustrated in Figure 7.1.

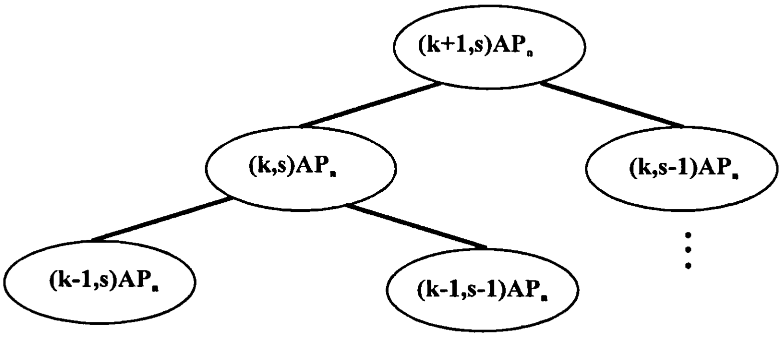


Figure 7.1: A hierarchy of assignment problems

The only exception is the case of $(k-1,1)AP_n$, whose solutions can always be extended to solutions of $(k,1)AP_n$. Observe that a solution to $(k-1,1)AP_n$ is a collection of n disjoint $(k-1)$-tuples, whereas a solution to $(k,1)AP_n$ requires a collection of n disjoint k-tuples. Hence, given a solution of $(k-1,1)AP_n$, the introduction of an additional n-set M_k, and the augmentation of each of $(k-1)$-tuples of the solution by a different element $m^k \in M_k$ yield a solution of the $(k,1)AP_n$. This observation has obvious algorithmic implications; instead of solving the $(k,1)AP_n$ directly (especially for large values of k), one can solve a lower dimensional axial assignment problem $(k',1)AP_n$, for $k' << k$, and, consequently, extend the solution to that of the higher dimensional problem. This algorithmic aspect is exploited in [73], within a scheme based on Lagrangean relaxation. It also appears in [12], the authors of which propose simple heuristics, which extend a solution of $(2,1)AP_n$ to a solution of $(k,1)AP_n$ by sequentially solving $(k-1)$ bipartite matchings. A *branch-and-bound* scheme, exploiting this feature, for the $(3,1)AP_n$ is presented in [11]. The principle is also applicable for $s \geq 2$, the difference being that solutions of $(k-1,s)AP_n$ might not be extendable to solutions of $(k,s)AP_n$. A simple case is that of pairs of *MOLS*, i.e. solutions to $(4,2)AP_n$, which are not always extendable to triples of *MOLS*, i.e. solutions of $(5,2)AP_n$. Another such case is the algorithm presented in [63], where an upper-bound heuristic for $(3,2)AP_n$, based on the same principle, is described.

A statement summarising the above, in terms of polyhedra, is the following.

Remark 7.2 *For $s \geq 2$,*

$$\begin{array}{rcl} proj_{x\in P_I^{(k-1,1)}}(P_I^{(k,1)}) & = & P_I^{(k-1,1)} \\ proj_{x\in P_I^{(k-1,s)}}(P_I^{(k,s)}) & \subseteq & P_I^{(k-1,s)} \end{array}$$

7.5 The axial assignment polytopes

The two most prominent representatives of this class are the polytopes $P_I^{(2,1)}$and $P_I^{(3,1)}$. We have seen that $dimP_I^{(2,1)} = (n-1)^2$. Other basic properties of $P_I^{(2,1)}$ can be found in [22, Theorem 1.2]. The facial structure of $P_I^{(3,1)}$ has also been substantially studied. As mentioned earlier, the dimension of this polytope is established, independently, in [9, 37]. Several classes of facets, induced by cliques and odd holes of the underlying intersection graph, are identified in [9, 76]. Separation algorithms for some of these classes are given in [9].

To the best of our knowledge, $(2,1)AP_n$ and $(3,1)AP_n$ are the only axial assignment problems, whose underlying polyhedral structure has been studied. However, several applications of axial assignment problems for $k > 3$, have been reported (see [22, 71]). This suggests that the study of $P_I^{(k,1)}$, for general k, is of both practical and theoretical interest. We have already mentioned that $P_I^{(k,1)} \neq \emptyset$, $\forall k,n \in \mathbb{Z}_+$. Further, it can be proved, by induction on n, that the number of vertices of $P_I^{(k,1)}$ is equal to $(n!)^{k-1}$. The proof is a simple extension of [9, Theorem 1.1]. Next, we establish the dimension of $P_I^{(k,1)}$, thus unifying and generalising the corresponding results obtained for $P_I^{(2,1)}$ and $P_I^{(3,1)}$. Recall from Corollary 7.7 that the dimension of the polytope associated with the linear relaxation of $P_I^{(k,1)}$, i.e. polytope $P^{(k-1)}$, is $dimP^{(k,1)} = \sum_{r=0}^{k-2}\binom{k}{r}\cdot(n-1)^{k-r}$.

Theorem 7.9 *For $n \geq 3$, $dimP_I^{(k,1)} = \sum_{r=0}^{k-2}\binom{k}{r}\cdot(n-1)^{k-r}$.*

Proof. We know that $\dim(P_I^{k,1}) \leq \dim(P^{k,1})$, since $P_I^{(k,1)} \subset P^{(k,1)}$. Strict inequality holds if and only if there exists an equality $ax = a_0$ satisfied by all $x \in P_I^{(k,1)}$, which cannot be expressed as a linear combination of the equality constraints $A_n^{(k,1)}x = e$. If we prove that no such equality exists, we essentially prove that the system $A_n^{(k,1)}x = e$ is the minimum equality system for both $P^{(k,1)}$ and $P_I^{(k,1)}$. Therefore, $\dim(P_I^{(k,1)}) = \dim(P^{(k,1)})$.

Assume that every $x \in P_I^{(k,1)}$ satisfies the equality $ax = a_0$, $a \in R^{n^k}$, $a_0 \in R$. Then, we show below that there exist scalars $\lambda_j^t, t = 1,..,k$, $j = 1,..,n$, such that:

$$a_{m^1...m^k} = \sum_{t=1}^{k} \lambda_{m^t}^t, \forall m^t \in M_t, t \in K \tag{7.19}$$

$$a_0 = \sum \left\{ \left[\sum \{\lambda_{m^t}^t : m^t \in M_t\} \right] : t \in K \right\} \tag{7.20}$$

Define:
$\lambda_{m^k}^1 = a_{1...1m^k}$.
$\lambda_{m^{k-1}}^2 = a_{1...m^{k-1}1} - a_{11...1}$,
$\vdots$
$\lambda_{m^2}^{k-1} = a_{1m^21...1} - a_{11...1}$,
$\lambda_{m^1}^k = a_{m^11...1} - a_{11...1}$
By substituting in (7.19) we get:

$$a_{m^1...m^k} = a_{m^11...1} + a_{1m^2...1} + ... + a_{1...1m^k} - (k-1)a_{11...1} \tag{7.21}$$

We will prove (7.21) by examining a number of collectively exhaustive cases. The proof essentially proceeds by induction on the number of indices that are different from 1. Also note that all the points used throughout the proof can be constructed by applying a series of index interchanges on the point $\widehat{x} \in P_I^{(k,1)}$, defined as:
$\widehat{x} = \left\{ \begin{array}{ll} \widehat{x}_{m^1...m^k} = 1, & \text{if } m^1 = m^2 = ... = m^k \\ \widehat{x}_{m^1...m^k} = 0, & \text{otherwise} \end{array} \right\}$
No more than three distinct values will be assumed for each index, hence the proof is valid for $n \geq 3$.

Case 7.9.1 *All indices are equal to* 1.

Equation (7.21) becomes a tautology.

Case 7.9.2 *Exactly one of the indices is different from* 1.

W.l.o.g. assume the case of $a_{1...1m^k}$ *all other cases being symmetrical. Substitution alone proves (7.21).*

Case 7.9.3 *Exactly two of the indices are different from* 1.

W.l.o.g. assume the case of $a_{1...1m^{k-1}m^k}$, *all other cases being symmetrical. Equation (7.21) can be written as:*

$$a_{1...1m^{k-1}m^k} = a_{1...1m^{k-1}1} + a_{1...1m^k} - a_{11...1}, \tag{7.22}$$

Consider a point $x \in P_I^{(k,1)}$, *such that* $x_{1...1} = x_{m^1m^2...m_s^{k-1}m^k} = 1$, *where:*
$s \in M^{k-1}$, $m^t \neq 1$ *for all* $t \in K$, $m_s^{k-1} \neq 1$, $m_s^{k-1} \neq m^{k-1}$.
Note that $x = \widehat{x}(m^1 \neq m^2?m^1 \leftrightarrow m^2)_2....(m^1 \neq m_s^{k-1}?m^1 \leftrightarrow m_s^{k-1})_{k-1}(m^1 \neq m^k?m^1 \leftrightarrow m^k)_k$.

Define the point $x' = x(1 \leftrightarrow m^k)_k$*. This implies that*

$$x'_{1...1} = x'_{m^1m^2...m_s^{k-1}m^k} = 0, x'_{1..1.m^k} = x'_{m^1m^2...m_s^{k-1}1} = 1$$

whereas $x' = x$ *in all other vector positions. Since* $x, x' \in P_I^{(k,1)}$*, they must both satisfy the equality* $ax = a_0$*, i.e.* $ax = ax'$*. Cancelling out terms on both sides results in:*

$$a_{11...1} + a_{m^1m^2...m_s^{k-1}m^k} = a_{1...1m^k} + a_{m^1m^2...m_s^{k-1}1} \tag{7.23}$$

Consider also a point $\overline{x} \in P_I^{(k,1)}$*, such that* $\overline{x}_{11...1m^{k-1}m^k} = \overline{x}_{m^1m^2...m_s^{k-1}1} = 1$*. This point can be derived from point* $\widehat{x}$ *as follows:*

(i) Let $\widehat{x}' = \widehat{x}(1 \leftrightarrow m^{k-1})_{k-1}(1 \leftrightarrow m^k)_k$ *. Observe that* $\widehat{x}'_{11...1m^{k-1}m^k} = 1$.

(ii) Let $\overline{x} = \widehat{x}'(m^1 \neq m^2?m^1 \leftrightarrow m^2)_2...(m^1 \neq m_s^{k-1}?m^1 \leftrightarrow m_s^{k-1})_{k-1}$*. Observe that* $\overline{x}_{m^1m^2...m_s^{k-1}1}$ $= 1$ *and also* $\overline{x}_{11...1m^{k-1}m^k} = 1$*, since* $m^1, m^2, ..., m^{k-2}, m_s^{k-1}$ *are different from* 1 *and* $m_s^{k-1} \neq m^{k-1}$.

Define the point $\overline{x}' = \overline{x}(m^k \leftrightarrow 1)_k$*. Clearly,*

$$\overline{x}'_{11...1m^{k-1}m^k} = \overline{x}'_{m^1m^2...m_s^{k-1}1} = 0, \overline{x}'_{11...1m^{k-1}1} = \overline{x}'_{m^1m^2...m_s^{k-1}m^k} = 1$$

and $x' = x$ *in all other positions. Both points belong to the polytope* $P_I^{(k,1)}$*, therefore* $a\overline{x} = a\overline{x}'$*. Cancelling out terms gives us:*

$$a_{1...1m^{k-1}m^k} + a_{m^1m^2...m_s^{k-1}1} = a_{11...1m^{k-1}1} + a_{m^1m^2...m_s^{k-1}m^k} \tag{7.24}$$

Adding equations (7.23) and (7.24) and cancelling terms results in:

$$a_{11...1} + a_{1...1m^{k-1}m^k} = a_{11...1m^{k-1}1} + a_{1...1m^k}$$

or

$$a_{1...1m^{k-1}m^k} = a_{1...1m^{k-1}1} + a_{1...1m^k} - a_{11...1},$$

which is equation (7.22).

Case 7.9.4 *For* $2 < r \leq k$*, exactly* r *of the indices are different from* 1.

W.l.o.g. assume the case of $a_{1...1m^{k-r+1}m^k}$*, all other cases being symmetrical. Equation (7.21) can be written as:*

$$a_{1..1m^{k-r+1}..m^k} = a_{1..1m^{k-r+1}1..1} + a_{1..1m^{k-r+2}1..1} + ... + a_{1...1m^k} - (r-1) \cdot a_{11..1} \tag{7.25}$$

The proof in this case constitutes the actual step of the induction. By the hypothesis of the induction, (7.21) is true if exactly $r - 1$ *of the indices are different from* 1*. This implies that the following equation holds.*

$$a_{1..1m^{k-r+2}..m^k} = a_{1..1m^{k-r+2}1..1} + a_{1..1m^{k-r+3}1..1} + ... + a_{1...1m^k} - (r-2) \cdot a_{11..1} \tag{7.26}$$

Using (7.26), equation (7.25) can be equivalently re-written as

$$a_{1..1m^{k-r+1}..m^k} = a_{1..1m^{k-r+1}1..1} + [a_{1..1m^{k-r+2}1..1} + ... + a_{1...1m^k} - (r-2) \cdot a_{11..1}] - a_{11..1}$$

or

$$a_{1..1m^{k-r+1}..m^k} = a_{1..1m^{k-r+1}1..1} + a_{1..1m^{k-r}..m^k} - a_{11..1}$$

or

$$a_{1..1m^{k-r+1}..m^k} + a_{11..1} = a_{1..1m^{k-r+1}1..1} + a_{1..1m^{k-r}..m^k} \tag{7.27}$$

Consider a point $x \in P_I^{(k,1)}$, *such that* $x_{1...1} = x_{m^1m^2..m_s^{k-r+1}m^{k-r+2}..m^k} = 1$, $m^t \neq 1$ *for all* $t \in K$, $m_s^{k-r+1} \neq 1$, $m_s^{k-r+1} \neq m^{k-r+1}$. *This point can be derived from* $\widehat{x}$ *following a variant of the interchanges used in Case 7.9.3.*

Define the point $x' = x(1 \leftrightarrow m^{k-r+2})_{k-r+2}...(1 \leftrightarrow m^k)_k$. *One can verify that*

$$x'_{1...1} = x'_{m^1..m_s^{k-r+1}..m^k} = 0, x'_{1..1m^{k-r+2}..m^k} = x'_{m^1...m_s^{k-r+1}1..1} = 1$$

Since both points belong to $P_I^{(k,1)}$, *the equality* $ax = ax'$ *holds. Cancelling terms gives us:*

$$a_{1...1} + a_{m^1m^2..m_s^{k-r+1}m^{k-r+2}..m^k} = a_{1..1m^{k-r+2}..m^k} + a_{m^1...m_s^{k-r+1}1..1} \tag{7.28}$$

Consider another point $\overline{x} \in P_I^{(k,1)}$, *such that* $\overline{x}_{m^1m^2..m_s^{k-r+1}1..1} = \overline{x}_{1..1m^{k-r+1}..m^k} = 1$ *(also derivable from* $\widehat{x}$*). Define the point* $\overline{x}' \in P_I^{(k,1)}$, *such that* $\overline{x}' = \overline{x}(1 \leftrightarrow m^{k-r+2})_{k-r+2}...(1 \leftrightarrow m^k)_k$. *Observe that*

$$\overline{x}'_{m^1m^2..m_s^{k-r+1}1..1} = \overline{x}'_{1..1m^{k-r+1}..m^k} = 0, \overline{x}'_{m^1m^2..m_s^{k-r+1}m^{k-r+2}..m^k} = \overline{x}'_{1..1m^{k-r+1}1..1} = 1$$

Again, the equality $a\overline{x} = a\overline{x}'$ *is valid and results in:*

$$a_{m^1m^2..m_s^{k-r+1}1..1} + a_{1..1m^{k-r+1}..m^k} = a_{m^1m^2..m_s^{k-r+1}m^{k-r+2}..m^k} + a_{1..1m^{k-r+1}1.1} \tag{7.29}$$

Combining equations (7.28) and (7.29) and cancelling out terms from both sides gives us equation (7.27). This completes the proof of equation (7.19).

Finally, since $\widehat{x} \in P_I^{(k,1)}$, $\widehat{x}$ must also satisfy the equation $ax = a_0$. Substituting the terms $a_{m^1...m^k}$ from (7.19) gives us the following:

$$a_0 = a_{1...1} + a_{2...2} + ... + a_{n...n}$$

or

$$a_0 = a_{1..1} + [a_{21..1} + .. + a_{1..12} - (k-1)a_{1..1}] + ... + [a_{n1..1} + .. + a_{1..1n} - (k-1)a_{1..1}]$$

or

$$\begin{aligned} a_0 \quad &= \quad [(a_{21..1} - a_{1..1}) + ... + (a_{n1..1} - a_{1..1})] + ... + [(a_{1..21} - a_{1..1}) + ... + (a_{1..n1} - a_{1..1})] \\ &\quad\; +(a_{1..1} + a_{1..12} + ... + a_{1..1n}) \end{aligned}$$

or

$$a_0 = \sum \left\{ \left[\sum \{ \lambda^t_{m^t} : m^t \in M_t \} \right] : t \in K \right\}$$

as required. ■

Let us examine which of the faces induced by the constraints of $P^{(k,1)}$ are facets of $P_I^{(k,1)}$. All the equality constraints (7.1) are satisfied by all points of $P_I^{(k,1)}$, therefore they define improper faces of

$P_I^{(k,1)}$. We show below that, for $c \in M_K$, the inequalities $x_c \geq 0$ define facets of $P_I^{(k,1)}$. In contrast, the inequalities $x_c \leq 1$ are redundant, i.e. implied by the original constraint set.

Proposition 7.10 *For $n \geq 3$, every inequality $x_c \geq 0$, for $c \in M_K$, defines a facet of $P_I^{(k,1)}$.*

Proof. For any $c \in M_K$, consider the polytope $P_I^{(k,1)c} = \{x \in P_I^{(k,1)} : x_c = 0\}$. It is sufficient to show that $\dim P_I^{(k,1)c} = \dim P_I^{(k,1)} - 1$. Evidently, $\dim P_I^{(k,1)c} \leq n^k - 1 - \operatorname{rank} A_n^{(k,1)c}$, where $A_n^{(k,1)c}$ is the matrix obtained from $A_n^{(k,1)}$ by removing column c. It is not difficult to see that the rank of $A_n^{(k,1)c}$ is equal to the rank of $A_n^{(k,1)}$. This is immediate if the column a^c is not among the columns of the upper triangular matrix described in Theorem 7.5, otherwise it follows by symmetry. Therefore, $\dim P_I^{(k,1)c} \leq \sum_{r=0}^{k-2} \binom{k}{r} \cdot (n-1)^{k-r} - 1$. To prove that this bound is attained, we use the same approach as in the proof of Theorem 7.9, i.e. show that any equation $ax = a_0$ (different than $x_c = 0$) satisfied for every $x \in P_I^{(k,1)c}$ is a linear combination of the system $A_n^{(k,1)c}x = e$. The proof goes through essentially unchanged. ■

Proposition 7.11 *For $n \geq 3$, every inequality $x_c \leq 1$ for $c \in M_K$ does not define a facet of $P_I^{(k,1)}$.*

Proof. For any $c \in M_K$ consider the polytope $P_I^{(k,1)c} = \{x \in P_I^{(k,1)} : x_c = 1\}$. We will show that $\dim P_I^{(k,1)c} < \dim P_I^{(k,1)} - 1$. We know already that $\dim P_I^{(k,1)c} < \dim P^{(k,1)c}$, where $P^{(k,1)c}$ is the LP-relaxation of $P_I^{(k,1)c}$. Setting x_c to one is equivalent to setting the variables, which appear at the same constraints with x_c, to zero. Note that a variable x_d appears at the same constraint with x_c, if and only if it has at least one index in common with x_c, i.e. if and only if $|c \cap d| \geq 1$. The number of variables x_d, such that $|c \cap d| = 0$ is $(n-1)^k$. It follows that the number of variables having at least one index in common with x_c, i.e. set to zero, if $x_c = 1$, is $n^k - (n-1)^k$.

Hence, $\dim P^{(k,1)c} = n^k - [n^k - (n-1)^k] - \operatorname{rank}(A_n^{(k,1)c})$ where $A_n^{(k,1)c}$ is the matrix obtained from $A_n^{(k,1)}$ by removing the columns corresponding to the variables set to zero. Obviously, $\operatorname{rank} A_n^{(k,1)c} \leq \operatorname{rank} A_n^{(k,1)}$. It follows that

$$\begin{aligned}\dim P^{(k,1)c} &\leq n^k - [n^k - (n-1)^k] - \operatorname{rank} A_n^{(k,1)} \\ &\leq \dim P_I^{(k,1)} - [n^k - (n-1)^k]\end{aligned}$$

Taking into account that $n^k - (n-1)^k > 1$ for $n \geq k \geq 3$, it holds that $\dim P^{(k,1)c} < \dim P_I^{(k,1)} - 1$. The result follows. ■

7.6 A family of facets for the axial assignment polytopes

Let C denote the index set of columns of the $0-1$ matrix A. We refer to a column of the A matrix as a^c for $c \in C$. The *intersection* graph $G_A = (V, E)$, has a node $c \in V$ for every $a^c \in A$, and an edge $(c_s, c_t) \in E$ for every pair of nodes with $a^{c_s} \cdot a^{c_t} \geq 1$. Let $G_A = (C^{(k,1)}, E^{(k,1)})$ denote the intersection graph of $A_n^{(k,1)}$. Then, $C^{(k,1)} \equiv M_K$ and $(c_s, c_t) \in E^{(k,1)}$ for all $c_s, c_t \in C^{(k,1)}$ with $|c_s \cap c_t| \geq 1$. By definition, $c \in C^{(k,1)}$ refers to the k-tuple $(m_c^1, \ldots, m_c^k) \in M_K$. Hence, the variable x_c is equivalently denoted as $x_{(m_c^1, \ldots, m_c^k)}$.

Proposition 7.12 *The graph $G_A = (C^{(k,1)}, E^{(k,1)})$ is regular of degree*

$$\textstyle\sum_{t=1}^{k-1} \binom{k}{t} (n-1)^{k-t}$$

Proof. For each $c_0 \in C^{(k,1)}$, there are exactly $(n-1)^k$ nodes with no index in common with c_0. Since c_0 is incident to all other nodes of G_A, the degree of node c_0 is $n^k - (n-1)^k - 1 = \sum_{t=0}^{k} \binom{k}{t}(n-1)^{k-t} - (n-1)^k - 1 = \sum_{t=1}^{k-1} \binom{k}{t}(n-1)^{k-t}$. ■

Corollary 7.13 $|E^{(k,1)}| = \frac{n^k(n^k-(n-1)^k-1)}{2}$.

Proof. The number of edges in a graph is the sum of the degrees of all nodes, divided by 2. ■

For $c_0 \in C^{(k,1)}$, define

$$Q^1(c_0) = \{c_t \in C^{(k,1)} : |c_0 \cap c_t| \geq \left\lfloor \frac{k}{2} \right\rfloor + 1\}$$

In other words, $Q^1(c_0)$ is the set of nodes having at least $\lfloor \frac{k}{2} \rfloor + 1$ indices in common with node c_0.

Definition 7.4 *Consider $c_0, c_t \in C^{(k,1)}$, with $|c_0 \cap c_t| \geq 1$. The complement of c_t with respect to c_0 is any element $\bar{c}_t(c_0) \in C^{(k,1)}$ such that $|c_t \cap \bar{c}_t(c_0)| = 0$, $|c_0 \cap c_t| + |c_0 \cap \bar{c}_t(c_0)| = k$.*

As an example, let $k = 4$ and $c_0 = (n, n, n, n)$, $c_t = (i_0, j_0, n, n)$. It is not difficult to see that $\bar{c}_t(c_0) = (n, n, k_0, l_0)$, where $i_0, j_0, k_0, l_0 \neq n$. Note that $(c_t, \bar{c}_t(c_0)) \notin E^{(k,1)}$.

For the rest of the section, assume $k \geq 3$. A *clique* is defined as a *maximal* complete subgraph.

Proposition 7.14 *For each $c_0 \in C^{(k,1)}$ and k odd, the node set $Q^1(c_0)$ induces a clique. There are n^k cliques of this type.*

Proof. Let $c_1, c_2 \in Q^1(c_0)$. Since both c_1, c_2 have at least $\lfloor \frac{k}{2} \rfloor + 1$ indices common with c_0, they must have at least one index in common with each other. It follows that $(c_1, c_2) \in E^{(k,1)}$. To show that $Q^1(c_0)$ is also maximal, consider $c_3 \in C^{(k,1)} \setminus Q^1(c_0)$. This implies $|c_0 \cap c_3| \leq \lfloor \frac{k}{2} \rfloor$. Because of k being odd, it holds that $|c_0 \cap \bar{c}_3(c_0)| \geq \lfloor \frac{k}{2} \rfloor + 1$, implying $\bar{c}_3(c_0) \in Q^1(c_0)$. Since $(c_3, \bar{c}_3(c_0)) \notin E^{(k,1)}$, the graph induced by $Q^1(c_0) \cup \{c_3\}$ is not complete, i.e. a contradiction. There is a distinct clique of this type for each node of $C^{(k,1)}$, i.e. the total number of cliques is n^k. ■

Remark 7.3 *Observe that $|Q^1(c_0)| = \sum_{t=\lfloor \frac{k}{2} \rfloor+1}^{k} \binom{k}{t} \cdot (n-1)^{k-t}$.*

For $k \geq 6$ and even, it can be verified that $Q^1(c_0)$ is not maximal, i.e. it has to be augmented by introducing additional nodes. Consider the example of $k = 6$ and let $c_0 = (n, n, n, n, n, n)$. Given that $\lfloor \frac{k}{2} \rfloor + 1 = 4$, the node set $Q^1(c_0)$ includes, apart from c_0, nodes having 4 or 5 indices in common with c_0. Hence:

$$\begin{aligned} Q^1(c_0) &= \{(m^1, n, n, n, n, n), ..., (n, n, n, n, n, m^6)\} \cup \\ &\quad \{(m^1, m^2, n, n, n, n), ..., (n, n, n, n, m^5, m^6)\} \cup \\ &\quad \{(n, n, n, n, n, n)\} \end{aligned}$$

where $m^1, ..., m^6 \in \{1, ..., n\}$. Notice that node $c_t = (m^1, m^2, m^3, n, n, n)$ is also connected to all nodes in $Q^1(c_0)$. Therefore, $Q^1(c_0)$ is not maximal. Nevertheless, it is not possible to include all nodes having 3 indices in common with c_0, since, for example nodes (m^1, m^2, m^3, n, n, n) and (n, n, n, m^4, m^5, m^6) are not connected. Observe that $(n, n, n, m^4, m^5, m^6) = \bar{c}_t(c_0)$.

To describe the set of nodes added to $Q^1(c_0)$, a number of intermediate definitions are necessary. For $c_0 \in C^{(k,1)}$ and $S \subset K$, define:

$$C_S^{(k,1)}(c_0) = \{c_t \in C^{(k,1)} : (c_0 \cap c_t) \in M_S\}$$

It is easy to verify that $|c_t \cap c_u| = 0$, for all $c_t \in C_S^{(k,1)}(c_0)$, $c_u \in C_{K\backslash S}^{(k,1)}(c_0)$. For k even, define $G = \{S \subset K : |S| = \frac{k}{2}\}$, $|G| = \binom{k}{\frac{k}{2}}$. Observe that, for k even, $S \in G$ if and only if $K\backslash S \in G$. Therefore, the set G can be partitioned into sets G^+, G^-, such that $G = G^+ \cup G^-$ and $S \in G^+$ if and only if $K\backslash S \in G^-$. There are $2^{\frac{|G|}{2}}$ such partitions. Define, finally, the set of nodes

$$Q^2_{G^+}(c_0) = \bigcup_{S \in G^+} C_S^{(k,1)}(c_0)$$

The node set $Q^2_{G^-}(c_0)$ is defined analogously. It follows that $c_t \in Q^2_{G^+}(c_0)$ if and only if $\bar{c}_t(c_0) \in Q^2_{G^-}(c_0)$.

Proposition 7.15 *For each $c_0 \in C^{(k,1)}$ and k even, the node set $Q^{1,2}(c_0) = Q^1(c_0) \cup Q^2_{G^+}(c_0)$ induces a clique.*

Proof. Let $c_1, c_2 \in Q^{1,2}(c_0)$. If both $c_1, c_2 \in Q^1(c_0)$ or both $c_1, c_2 \in Q^2_{G^+}(c_0)$, it is easy to verify that $|c_1 \cap c_2| \geq 1$, i.e. $(c_1, c_2) \in E^{(k,1)}$. If $c_1 \in Q^1(c_0)$, $c_2 \in Q^2_{G^+}(c_0)$, c_1 has $\frac{k}{2} + 1$ indices in common with c_0, and c_2 has $\frac{k}{2}$ indices in common with c_0. Because of k being even, it follows that c_1, c_2 have at least one index in common.

To show that $Q^{1,2}(c_0)$ is maximal, consider $c_3 \in C^{(k,1)} \setminus Q^{1,2}(c_0)$ and note that either $|c_3 \cap c_0| \leq \frac{k}{2} - 1$ or $|c_3 \cap c_0| = \frac{k}{2}$. The first case implies that $|\bar{c}_3(c_0) \cap c_0| \geq \frac{k}{2} + 1$, i.e. $\bar{c}_3(c_0) \in Q^1(c_0)$. Hence, the graph induced by $Q^{1,2}(c_0) \cup \{c_3\}$ is not complete. In the second case, clearly, $c_3 \notin Q^2_{G^+}(c_0)$. Hence, there exists at least one element of $Q^2_{G^+}(c_0)$, namely c_4, such that $|c_3 \cap c_4| = 0$, i.e. the graph induced by $Q^{1,2}(c_0) \cup \{c_3\}$ is not complete. ∎

For k even, it can be verified that:

$$|Q^{1,2}(c_0)| = \sum_{t=\frac{k}{2}+1}^{k} \binom{k}{t} \cdot (n-1)^{k-t} + \frac{1}{2} \cdot \binom{k}{\frac{k}{2}} \cdot (n-1)^{\frac{k}{2}}$$

Overall, the node set of cliques belonging to this family is:

$$Q(c) = \left\{ \begin{array}{ll} Q^1(c) & : k \text{ odd} \\ Q^{1,2}(c) & : k \text{ even} \end{array} \right\}$$

Theorem 7.16 *For $k \geq 3$, $n \geq 4$, the inequality*

$$\sum \{x_q : q \in Q(c)\} \leq 1 \tag{7.30}$$

defines a facet of $P_I^{(k,1)}$, for every $c \in C^{(k,1)}$.

The proof of the following theorem can be found in [3]. Note that this class of facets induced by clique inequalities is a generalisation of a family of facets in [9].

7.7 The planar assignment polytopes

Each integer point of $P_I^{(3,2)}$ corresponds to a Latin square of order n ([37]). The polytope $P_I^{(3,2)}$ is also referred to as *the Latin square* polytope. In general, $P_I^{(k,2)}$ can be called the $(k-2)$*MOLS* polytope. The connection between *MOLS* and $P_I^{(k,2)}$ provides information on the non-emptiness of $P_I^{(k,2)}$ for certain values of the parameters k, n. An immediate result is that $P_I^{(3,2)} \neq \emptyset$, $\forall n \in \mathbb{Z}_+ \backslash \{1\}$, since there exist more than one Latin square for any $n \geq 2$. Further, it is known that there cannot be more than $n-1$ *MOLS* of order n ([60, Theorem 2.1]). This implies that $k-2 \leq n-1$ or $k \leq n+1$ is a necessary condition for $P_I^{(k,2)} \neq \emptyset$. Observe that this is a special case of Proposition 7.8. The theory of *MOLS* provides us with further results, not implied by Proposition 7.8. For example, it is known that $P_I^{(4,2)} \neq \emptyset$, $\forall n \in \mathbb{Z}_+ \setminus \{1,2,6\}$ ([60, Theorem 2.9]).

The facial structure of $P_I^{(3,2)}$ and $P_I^{(4,2)}$ have been studied in [37] and [2] respectively. The following theorem unifies and generalises the results of both studies concerning the dimension.

Theorem 7.17 *For* $n \geq 5$, $k < n+1$, *if* $P_I^{(k,2)} \neq \emptyset$ *then* $\dim(P_I^{(k,2)}) = \sum_{r=0}^{k-2}\left[\binom{k}{r}\cdot(n-1)^{k-r}\right]$

Proof. Since $P_I^{(k,2)} \subset P^{(k,2)}$, it follows that $dimP_I^{(k,2)}) \leq dim(P^{(k,2)})$, strict inequality holding if and only if there exists an equality $ax = a_0$ satisfied by all $x \in P_I^{(k,2)}$, which cannot be expressed as a linear combination of the equality constraints $A_n^{(k,2)}x = e$. Proving that no such equality exists, implies that the system $A_n^{(k,2)}x = e$ is the minimum equality system for both $P^{(k,2)}$ and P. It follows that $dim(P_I^{(k,2)}) = dim(P^{(k,2)})$.

Assume that every $x \in P_I^{(k,2)}$ satisfies the equality $ax = a_0$, $a \in R^{n^k}$, $a_0 \in R$. Then, there exist scalars $\lambda^1_{m^{k-1}m^k}$, $\lambda^2_{m^{k-2}m^k}$, $\lambda^3_{m^{k-2}m^{k-1}}, \ldots, \lambda^{\frac{k(k-1)}{2}}_{m^1m^2}$, such that:

$$a_{m^1\ldots m^k} = \lambda^1_{m^{k-1}m^k} + \lambda^2_{m^{k-2}m^k} + \lambda^3_{m^{k-2}m^{k-1}} + \ldots + \lambda^{\frac{k(k-1)}{2}}_{m^1m^2} \qquad (7.31)$$

$$a_0 = \sum_{(m^{k-1},m^k)\in M_{k-1}\times M_k} \lambda^1_{m^{k-1}m^k} + \ldots + \sum_{(m^1,m^2)\in M_1\times M_2} \lambda^{\frac{k(k-1)}{2}}_{m^1m^2} \qquad (7.32)$$

Define:

$$\begin{array}{lcl}
\lambda^1_{m^{k-1}m^k} & = & a_{1\ldots 1m^{k-1}m^k} \\
\lambda^2_{m^{k-2}m^k} & = & a_{1\ldots 1m^{k-2}1m^k} - a_{1..1m^k} \\
\lambda^3_{m^{k-2}m^{k-1}} & = & a_{1\ldots 1m^{k-2}m^{k-1}1} - a_{1..1m^{k-2}11} - a_{1..1m^{k-1}1} + a_{1..11} \\
\lambda^4_{m^{k-3}m^k} & = & a_{1\ldots 1m^{k-3}11m^k} - a_{1..1m^k} \\
\lambda^5_{m^{k-3}m^{k-2}} & = & a_{1\ldots 1m^{k-3}m^{k-2}11} - a_{1..1m^{k-3}111} - a_{1..1m^{k-2}11} + a_{1..11} \\
\vdots & & \\
\lambda^{\frac{k(k-1)}{2}}_{m^1m^2} & = & a_{m^1m^21\ldots 11} - a_{m^11..11} - a_{1m^21..11} + a_{1..11}
\end{array}$$

Observe the difference in the expression of $\lambda^t_{m^im^j}, t, i, j \in K$, $i < j$, when $j = k$ and $j \neq k$. For $j = k$, only n rows of the corresponding row set (m^s, m^k), $1 \leq s < k$, are assigned a zero weight, whereas for $j = k$, exactly $2n-1$ rows are assigned a zero weight. These rows are the ones removed as linearly dependent in the proof of Proposition 7.5.

By substituting the values of scalars in equation (7.31), we obtain:

$$\begin{aligned}
a_{m^1\ldots m^k} = {} & a_{m^1m^21..1} + a_{m^11m^31..1} + \ldots + a_{1..1m^{k-2}1m^k} + a_{1..1m^{k-1}m^k} \\
& -(k-2)\cdot(a_{m^11..1} + a_{1m^21..1} + \ldots + a_{1..1m^{k-1}1} + a_{1..1m^k}) + \\
& +\frac{(k-1)\cdot(k-2)}{2}a_{11..1}
\end{aligned} \qquad (7.33)$$

Before we proceed, we have to use an intermediate result. A point $x \in P_I^{(k,2)}$ represents a set of $k-2$ MOLS of order n. Assume w.l.o.g. that sets M_1 and M_2 index the rows and the columns of the Latin squares respectively. The values of Latin square $L_t, t = 1,..,k-2$, belong to set M_{t+2}. The content of cell (m^1, m^2) at Latin square L_t at point x can be denoted as $m^{t+2}(m^1, m^2)_x$. The following proposition is simply the generalisation of Proposition 3.6.

Proposition 7.18 *For $n \geq 5$, $k < n+1$, if $P_I^{k,n} \neq \varnothing$ the following equation holds.*

$$\begin{aligned}
& a_{m_1^1 m_1^2 m^3(m_1^1,m_1^2)..m_1^k(m_1^1,m_1^2)} + a_{m_1^1 m_1^2 m^3(m_2^1,m_2^2)..m^k(m_2^1,m_2^2)} + \\
& a_{m_1^1 m_2^2 m^3(m_1^1,m_2^2)..m_1^k(m_1^1,m_2^2)} + a_{m_1^1 m_2^2 m^3(m_2^1,m_1^2)..m^k(m_2^1,m_1^2)} + \\
& a_{m_2^1 m_1^2 m^3(m_1^1,m_2^2)..m_1^k(m_1^1,m_2^2)} + a_{m_2^1 m_1^2 m^3(m_2^1,m_1^2)..m^k(m_2^1,m_1^2)} + \\
& a_{m_2^1 m_2^2 m^3(m_1^1,m_1^2)..m_1^k(m_1^1,m_1^2)} + a_{m_2^1 m_2^2 m^3(m_2^1,m_2^2)..m^k(m_2^1,m_2^2)} \\
= \; & a_{m_1^1 m_1^2 m^3(m_1^1,m_2^2)..m_1^k(m_1^1,m_2^2)} + a_{m_1^1 m_1^2 m^3(m_2^1,m_1^2)..m^k(m_2^1,m_1^2)} + \\
& a_{m_1^1 m_2^2 m^3(m_1^1,m_1^2)..m_1^k(m_1^1,m_1^2)} + a_{m_1^1 m_2^2 m^3(m_2^1,m_2^2)..m^k(m_2^1,m_2^2)} + \\
& a_{m_2^1 m_1^2 m^3(m_1^1,m_1^2)..m_1^k(m_1^1,m_1^2)} + a_{m_2^1 m_1^2 m^3(m_2^1,m_2^2)..m^k(m_2^1,m_2^2)} + \\
& a_{m_2^1 m_2^2 m^3(m_1^1,m_2^2)..m_1^k(m_1^1,m_2^2)} + a_{m_2^1 m_2^2 m^3(m_2^1,m_1^2)..m^k(m_2^1,m_1^2)} \qquad (7.34)
\end{aligned}$$

for all $m_1^1, m_2^1 \in M_1$, $m_1^2, m_2^2 \in M_2$, $m_1^1 \neq m_2^1$, $m_1^2 \neq m_2^2$.

Proof. Assuming $P_I^{(k,2)} \neq \varnothing$, there exists an arbitrary point $x_1 \in P_I^{(k,2)}$, having four specific variables set to 1, as illustrated in Table 7.4, where $m_i^t \in M_t$, $i = 1,..,4$, $t \in \{3,...,k\}$. Existence of such a point allows for the points $x_2 = x_1(m_1^1 \leftrightarrow m_2^1)_1$, $x_3 = x(m_1^2 \leftrightarrow m_2^2)_2$, $x_4 = x_3(m_1^1 \leftrightarrow m_2^1)_1$ to be derived. Tables 7.5, 7.6 and 7.7 depict these points.

Table 7.4: Point x_1 (Proposition 7.18)

	$\cdots$	m_1^2	$\cdots$	m_2^2	$\cdots$
$\vdots$					
m_1^1		$m_1^3...m_1^k$		$m_2^3...m_2^k$	
$\vdots$					
m_2^1		$m_3^3...m_3^k$		$m_4^3...m_4^k$	
$\vdots$					

Table 7.5: Point x_2 (Proposition 7.18)

	$\cdots$	m_1^2	$\cdots$	m_2^2	$\cdots$
$\vdots$					
m_1^1		$m_3^3...m_3^k$		$m_4^3...m_4^k$	
$\vdots$					
m_2^1		$m_1^3...m_1^k$		$m_2^3...m_2^k$	
$\vdots$					

Table 7.6: Point x_3 (Proposition 7.18)

	$\cdots$	m_1^2	$\cdots$	m_2^2	$\cdots$
$\vdots$					
m_1^1		$m_2^3...m_2^k$		$m_1^3...m_1^k$	
$\vdots$					
m_2^1		$m_4^3...m_4^k$		$m_3^3...m_3^k$	
$\vdots$					

Table 7.7: Point x_4 (Proposition 7.18)

	$\cdots$	m_1^2	$\cdots$	m_2^2	$\cdots$
$\vdots$					
m_1^1		$m_4^3...m_4^k$		$m_3^3...m_3^k$	
$\vdots$					
m_2^1		$m_2^3...m_2^k$		$m_1^3...m_1^k$	
$\vdots$					

Since $x_1, x_2 \in P_I^{(k,2)}$, the equation $ax_1 = ax_2$ holds. Observe that the two points have exactly the same non-zero variables, except for variables corresponding to cells at rows m_1^1 and m_2^1, i.e. variables of the form $x_{m_i^1 m^2..m^k}$, $i = 1, 2$. Cancelling those identical terms on both sides of the equation, we get:

$$\begin{aligned}
& a_{m_1^1 m_1^2 m_1^3..m_1^k} + a_{m_1^1 m_2^2 m_2^3..m_2^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_1^1 m^2 m^3(m_1^1, m^2)_{x_1}..m^k(m_1^1, m^2)_{x_1}} + \\
& a_{m_2^1 m_1^2 m_3^3..m_3^k} + a_{m_2^1 m_2^2 m_4^3..m_4^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_2^1 m^2 m^3(m_2^1, m^2)_{x_1}..m^k(m_2^1, m^2)_{x_1}} \\
= \; & a_{m_1^1 m_1^2 m_3^3..m_3^k} + a_{m_1^1 m_2^2 m_4^3..m_4^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_1^1 m^2 m^3(m_2^1, m^2)_{x_2}..m^k(m_2^1, m^2)_{x_2}} + \\
& a_{m_2^1 m_1^2 m_1^3..m_1^k} + a_{m_2^1 m_2^2 m_2^3..m_2^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_2^1 m^2 m^3(m_2^1, m^2)_{x_2}..m^k(m_2^1, m^2)_{x_2}} \qquad (7.35)
\end{aligned}$$

Since $x_3, x_4 \in P_I^{(k,2)}$, the equation $ax_3 = ax_4$ also holds. Again observe that the points x_3 & x_4 have exactly the same non-zero variables, except for variables corresponding to cells at rows m_1^1, m_2^1, i.e. variables $x_{m_i^1 m^2..m^k}$, $i = 1, 2$. Cancelling out those identical terms on both sides of the equation, we get:

$$\begin{aligned}
& a_{m_1^1 m_1^2 m_2^3..m_2^k} + a_{m_1^1 m_2^2 m_1^3..m_1^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_1^1 m^2 m^3(m_1^1, m^2)_{x_3}..m^k(m_1^1, m^2)_{x_3}} + \\
& a_{m_2^1 m_1^2 m_4^3..m_4^k} + a_{m_2^1 m_2^2 m_3^3..m_3^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_2^1 m^2 m^3(m_2^1, m^2)_{x_4}..m^k(m_2^1, m^2)_{x_3}} \\
= \; & a_{m_1^1 m_1^2 m_4^3..m_4^k} + a_{m_1^1 m_2^2 m_3^3..m_3^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_1^1 m^2 m^3(m_2^1, m^2)_{x_2}..m^k(m_2^1, m^2)_{x_4}} + \\
& a_{m_2^1 m_1^2 m_2^3..m_2^k} + a_{m_2^2 m_2^2 m_1^3..m_1^k} + \sum_{m^2 \in M_2 \backslash \{m_1^2, m_2^2\}} a_{m_2^1 m^2 m^3(m_2^1, m^2)_{x_2}..m^k(m_2^1, m^2)_{x_4}} \qquad (7.36)
\end{aligned}$$

The critical step of this proof is to observe that, for $m^2 \in M_2 \backslash \{m_1^2, m_2^2\}$, $t \in \{3, .., k\}$, the following equations hold.

(i) $m^t(m_1^1, m^2)_{x_1} = m^t(m_1^1, m^2)_{x_3}$, $m^t(m_2^1, m^2)_{x_1} = m^t(m_2^1, m^2)_{x_3}$

(ii) $m^t(m_1^1, m^2)_{x_2} = m^t(m_1^1, m^2)_{x_4}$, $m^t(m_2^1, m^2)_{x_2} = m^t(m_2^1, m^2)_{x_4}$

Table 7.8: Point x_0 (Theorem 7.17)

	1	$\cdots$	m_0^2	$\cdots$
1	11..1		$m_1^3 m_1^4 \dots m_1^k$	
$\vdots$				
m_0^1	$m_2^3 m_2^4 \dots m_2^k$		$m_3^3 m_3^4 \dots m_3^k$	
$\vdots$				

Table 7.9: Point x'_0 (Theorem 7.17)

	1	$\cdots$	m_0^2	$\cdots$
1	$m_0^3 m_0^4 \dots m_0^k$		$m_1^3 m_1^4 \dots m_1^k$	
$\vdots$				
m_0^1	$m_2^3 m_2^4 \dots m_2^k$		$m_3^3 m_3^4 \dots m_3^k$	
$\vdots$				

Therefore, subtracting equation (7.36) from (7.35) results in

$$\begin{aligned}
& a_{m_1^1 m_1^2 m_1^3 .. m_1^k} + a_{m_1^1 m_2^2 m_2^3 .. m_2^k} - a_{m_1^1 m_1^2 m_2^3 .. m_2^k} - a_{m_1^1 m_2^2 m_1^3 .. m_1^k} \\
+ \quad & a_{m_2^1 m_1^2 m_3^3 .. m_3^k} + a_{m_2^1 m_2^2 m_4^3 .. m_4^k} - a_{m_2^1 m_1^2 m_4^3 .. m_4^k} - a_{m_2^1 m_2^2 m_3^3 .. m_3^k} \\
= \quad & a_{m_1^1 m_1^2 m_3^3 .. m_3^k} + a_{m_1^1 m_2^2 m_4^3 .. m_4^k} - a_{m_1^1 m_1^2 m_4^3 .. m_4^k} - a_{m_1^1 m_2^2 m_3^3 .. m_3^k} \\
+ \quad & a_{m_2^1 m_1^2 m_1^3 .. m_1^k} + a_{m_2^1 m_2^2 m_2^3 .. m_2^k} - a_{m_2^1 m_1^2 m_2^3 .. m_2^k} - a_{m_2^2 m_2^2 m_1^3 .. m_1^k}
\end{aligned}$$

Changing the side of negated terms and substituting m_1^t, m_2^t, m_3^t, m_4^t by $m^t(m_1^1, m_1^2)$, $m^t(m_1^1, m_2^2)$, $m^t(m_2^1, m_1^2)$, $m^t(m_2^1, m_2^2)$, respectively, illustrates that the above equation is identical to (7.35). ∎

(Back to the proof of Theorem 7.17). To prove that (7.33) holds, we examine a number of collectively exhaustive cases and proceeds by induction on the number of indices $m^t, t \in K$, which are different from 1. The essential part of the proof, repeated for all cases, involves two steps. First, given that $P_I^{(k,2)}$ is non-empty, there exists a point x_0, as illustrated at Table 7.8. This point represents a set of $k-2$ MOLS. Each Latin square has element 1 at cell $(1,1)$. Moreover, at cell (m_0^1, m_0^2), square L_1 has element m_3^3, square L_2 has element m_3^4, etc. In general, for $t = 3, ..., k$, square L_{t-2}, has element m_1^t at cell $(1, m_0^2)$, element m_2^t at cell $(m_0^1, 1)$ and element m_3^t at cell (m_0^1, m_0^2). The second important aspect is that, by applying certain index interchanges, we can derive additional integer points of $P_I^{(k,2)}$. By hypothesis, all these points satisfy the equality $ax = a_0$.

We first prove the existence of point x_0. Given that $P_I^{(k,2)} \neq \emptyset$, a certain point x'_0 can be assumed to have the general form shown at Table 7.9. Consider the point:

$$x_0 = x'_0(m_0^3 \neq 1?m_0^3 \leftrightarrow 1)_3(m_0^4 \neq 1?m_0^4 \leftrightarrow 1)_4 \dots (m_0^k \neq 1?m_0^k \leftrightarrow 1)_k$$

and observe that it has exactly the form of Table 7.8.

Let us now present the steps of the induction.

Case 7.18.1 *All indices are equal to* 1.

Equation (7.33) becomes a tautology.

Case 7.18.2 *One or two of the indices is different from* 1.

Table 7.10: Point x_2 (Theorem 7.17, Case 7.18.3)

	1	$\cdots$	m_0^2	$\cdots$
1	$m_0^3 1 \ldots 1$		$m_1^3 \ldots m_1^{k-2} m_1^{k-1} m_1^k$	
$\vdots$				
m_0^1	$m_2^3 \ldots m_2^{k-2} m_2^{k-1} m_2^k$		$m_3^3 \ldots m_3^{k-2} m_3^{k-1} m_3^k$	
$\vdots$				

W.l.o.g., for $m_0^1, m_0^2 \neq 1$, assume the cases of $a_{m_0^1 1 \ldots 11}$ and $a_{m_0^1 m_0^2 1..1}$, respectively, all other cases being symmetrical. Equation (7.33) can be proved by substitution.

Case 7.18.3 *Exactly three of the indices are different from* 1.

W.l.o.g., for $m_0^1, m_0^2, m_0^3 \neq 1$, assume the case of $a_{m_0^1 m_0^2 m_0^3 1..1}$, all other cases being symmetrical. Equation (7.33) can be written as:

$$\begin{aligned} a_{m_0^1 m_0^2 m_0^3 1..1} &= a_{m_0^1 m_0^2 1..1} + a_{m_0^1 1 m_0^3 1..1} + a_{1 m_0^2 m_0^3 1..1} \\ &\quad - a_{m_0^1 1..1} - a_{1 m_0^2 1..1} - a_{11 m_0^3 1..1} \\ &\quad + a_{1..1} \end{aligned} \tag{7.37}$$

Assume point $x_1 = x_0$.

At point x_1, equation (7.34) is written as:

$$\begin{aligned} & a_{11\ldots1} + a_{11 m_3^3 \ldots m_3^k} + a_{1 m_0^2 m_1^3 \ldots m_1^k} + a_{1 m_0^2 m_2^3 \ldots m_2^k} + \\ & a_{m_0^1 1 m_1^3 \ldots m_1^k} + a_{m_0^1 1 m_2^3 \ldots m_2^k} + a_{m_0^1 m_0^2 1..1} + a_{m_0^1 m_0^2 m_3^3 \ldots m_3^k} \\ = \ & a_{11 m_1^3 \ldots m_1^k} + a_{11 m_2^3 \ldots m_2^k} + a_{1 m_0^2 1..1} + a_{1 m_0^2 m_3^3 \ldots m_3^k} + \\ & a_{m_0^1 11..1} + a_{m_0^1 1 m_3^3 \ldots m_3^k} + a_{m_0^1 m_0^2 m_1^3 \ldots m_1^k} + a_{m_0^1 m_0^2 m_2^3 \ldots m_2^k} \end{aligned} \tag{7.38}$$

Consider the elements m_3^3, m_3^4. Note that, since pair $(1,1)$ appears already at cell $(1,1)$ for the pair of Latin squares L_1, L_2, at most one of m_3^3, m_3^4 can be equal to 1. *Assume w.l.o.g. that $m_3^3 \neq 1$. If $m_3^3 = 1$, it must be that $m_3^4 \neq 1$ and we can simply interchange the roles of sets M_3, M_4.*

Derive the point $x_2 = x_1(1 \leftrightarrow m_0^3)_3$, illustrated at Table 7.10.

At point x_1, equation (7.34) is written as:

$$\begin{aligned} & a_{11 m_0^3 1..1} + a_{11 m_3^3 \ldots m_3^k} + a_{1 m_0^2 m_1^3 \ldots m_1^k} + a_{1 m_0^2 m_2^3 \ldots m_2^k} + \\ & a_{m_0^1 1 m_1^3 \ldots m_1^k} + a_{m_0^1 1 m_2^3 \ldots m_2^k} + a_{m_0^1 m_0^2 m_0^3 1..1} + a_{m_0^1 m_0^2 m_3^3 \ldots m_3^k} \\ = \ & a_{11 m_1^3 \ldots m_1^k} + a_{11 m_2^3 \ldots m_2^k} + a_{1 m_0^2 m_0^3 1..1} + a_{1 m_0^2 m_3^3 \ldots m_3^k} + \\ & a_{m_0^1 1 m_0^3 1..1} + a_{m_0^1 1 m_3^3 \ldots m_3^k} + a_{m_0^1 m_0^2 m_1^3 \ldots m_1^k} + a_{m_0^1 m_0^2 m_2^3 \ldots m_2^k} \end{aligned} \tag{7.39}$$

Subtracting equation (7.39) from equation (7.38) results in

$$\begin{aligned} & a_{11..1} - a_{11 m_0^3 1..1} + a_{m_0^1 m_0^2 1..1} - a_{m_0^1 m_0^2 m_0^3 1..1} \\ = \ & a_{1 m_0^2 1..1} - a_{1 m_0^2 m_0^3 1..1} + a_{m_0^1 1..1} - a_{m_0^1 1 m_0^3 1..1} \end{aligned}$$

Table 7.11: Point x_1 (Theorem 7.17, Case 7.18.4)

	1	$\cdots$	m_0^2	$\cdots$
1	$m_0^3..m_0^{r-1}1\dots1$		$m_1^3\dots m_1^{k-2}m_1^{k-1}m_1^k$	
$\vdots$				
m_0^1	$m_2^3\dots m_2^{k-2}m_2^{k-1}m_2^k$		$m_3^3\dots m_3^{k-2}m_3^{k-1}m_3^k$	
$\vdots$				

or

$$\begin{aligned}&a_{m_0^1m_0^2m_0^31..1} = a_{m_0^1m_0^21..1} + a_{m_0^11m_0^31..1} + a_{1m_0^2m_0^31..1}\\&-a_{m_0^11..1} - a_{1m_0^21..1} - a_{11m_0^31..1} + a_{11..1}\end{aligned}$$

which is equation (7.37).

Case 7.18.4 *For $3 < r \leq k$, exactly r of the indices are different from 1.*

W.l.o.g., for $m_0^1, m_0^2, ..., m_0^r \neq 1$, assume the case of $a_{m_0^1m_0^2...m_0^r1..1}$, all other cases being symmetrical. Equation (7.33) can be written as:

$$\begin{aligned}a_{m_0^1m_0^2m_0^31..1} \quad = \quad & a_{m_0^1m_0^21..1} + a_{m_0^11m_0^31..1} + ... + a_{1..1m_0^{r-1}m_0^r1..1}\\&-(r-2)\cdot(a_{m_0^11..1} + a_{1m_0^21..1} + ... + a_{1..1m_0^r1..1})\\&+\frac{(r-1)\cdot(r-2)}{2}a_{1..1}\end{aligned} \tag{7.40}$$

This case represents the actual step of the induction. Let us assume that equation (7.33) holds for all coefficients having exactly $r-1$ indices different from 1, therefore being true for coefficients $a_{m_0^1m_0^2...m_0^{r-1}1..1}$, $a_{1m_0^2m_0^3..m^r1..1}$, $a_{m_0^11m_0^3..m_0^r1..1}$. Equation (7.33) must also hold by hypothesis for all coefficients having exactly $r-2$ indices different from 1, including coefficients $a_{m_0^111m_0^4..m_0^r1..1}$, $a_{1m_0^21m_0^4..m_0^r1..1}$, $a_{11m_0^3m_0^4..m_0^r1..1}$. The same being true for coefficients with exactly $r-3$ indices different from 1, coefficient $a_{11m_0^3..m_0^{r-1}1..1}$ satisfies equation (7.33).

By substituting terms, it can be proved that the following equation is identical to (7.40).

$$\begin{aligned}a_{m_0^1m_0^2...m_0^r1..1} \quad = \quad & a_{m_0^1m_0^2...m_0^{r-1}1..1} + a_{1m_0^2m_0^3..m^r1..1} + a_{m_0^11m_0^3..m_0^r1..1}\\&-a_{m_0^111m_0^4..m_0^r1..1} - a_{1m_0^21m_0^4..m_0^r1..1} - a_{11m_0^3m_0^4..m_0^r1..1}\\&+a_{11m_0^3..m_0^{r-1}1..1}\end{aligned} \tag{7.41}$$

Assume point $x_1 = x_0(1 \leftrightarrow m_0^3)_3...(1 \leftrightarrow m_0^{r-1})_3$, depicted at Table 7.11.

At point x_1, equation (7.34) is written as:

$$\begin{aligned}& a_{11m_0^3..m_0^{r-1}1..1} + a_{11m_3^3...m_3^k} + a_{1m_0^2m_1^3...m_1^k} + a_{1m_0^2m_2^3...m_2^k} +\\& a_{m_0^11m_1^3...m_1^k} + a_{m_0^11m_2^3...m_2^k} + a_{m_0^1m_0^2m_0^3..m_0^{r-1}1..1} + a_{m_0^1m_0^2m_3^3...m_3^k}\\= \quad & a_{11m_1^3...m_1^k} + a_{11m_2^3...m_2^k} + a_{1m_0^2m_0^3..m_0^{r-1}1..1} + a_{1m_0^2m_3^3...m_3^k} +\\& a_{m_0^11m_0^3..m_0^{r-1}1..1} + a_{m_0^11m_3^3...m_3^k} + a_{m_0^1m_0^2m_1^3...m_1^k} + a_{m_0^1m_0^2m_2^3...m_2^k}\end{aligned} \tag{7.42}$$

Consider again the elements m_3^r, m_3^{r+1}. Note that, since pair $(1,1)$ appears already at cell $(1,1)$

Table 7.12: Point x_2 (Theorem 7.17, Case 7.18.4)

	1	$\cdots$	m_0^2	$\cdots$
1	$m_0^3..m_0^r1\dots1$		$m_1^3\dots m_1^{k-2}m_1^{k-1}m_1^k$	
$\vdots$				
m_0^1	$m_2^3\dots m_2^{k-2}m_2^{k-1}m_2^k$		$m_3^3\dots m_3^{k-2}m_3^{k-1}m_3^k$	
$\vdots$				

for the corresponding pair of Latin squares (i.e. L_{r-2}, L_{r-1}), at most one of m_3^r, m_3^{r+1} can be equal to 1. *Assume w.l.o.g. that $m_3^r \neq 1$. If $m_3^r = 1$, then certainly $m_3^{r+1} \neq 1$ and we can interchange the roles of sets M_r, M_{r+1}.*

Derive point $x_2 = x_1(1 \leftrightarrow m_0^r)_r$, illustrated at Table 7.12.

At point x_2, equation (7.34) is written as:

$$\begin{aligned} & a_{11m_0^3..m_0^r1..1} + a_{11m_3^3...m_3^k} + a_{1m_0^2m_1^3...m_1^k} + a_{1m_0^2m_2^3...m_2^k} + \\ & a_{m_0^11m_1^3...m_1^k} + a_{m_0^11m_2^3...m_2^k} + a_{m_0^1m_0^2m_0^3..m_0^r1..1} + a_{m_0^1m_0^2m_3^3...m_3^k} \\ = \;& a_{11m_1^3...m_1^k} + a_{11m_2^3...m_2^k} + a_{1m_0^2m_0^3..m_0^r1..1} + a_{1m_0^2m_3^3...m_3^k} + \\ & a_{m_0^11m_0^3..m_0^r1..1} + a_{m_0^11m_3^3...m_3^k} + a_{m_0^1m_0^2m_1^3...m_1^k} + a_{m_0^1m_0^2m_2^3...m_2^k} \end{aligned} \tag{7.43}$$

Subtracting equation (7.43) from equation (7.42) results in

$$\begin{aligned} & a_{11m_0^3..m_0^{r-1}1..1} - a_{11m_0^3..m_0^r1..1} + a_{m_0^1m_0^2m_0^3..m_0^{r-1}1..1} - a_{m_0^1m_0^2m_0^3..m_0^r1..1} \\ = \;& a_{1m_0^2m_0^3..m_0^{r-1}1..1} - a_{1m_0^2m_0^3..m_0^r1..1} + a_{m_0^11m_0^3..m_0^{r-1}1..1} - a_{m_0^11m_0^3..m_0^r1..1} \end{aligned}$$

which is equation (7.41).

Up to this point, equation (7.33) and its equivalence, equation (7.31), have been proved. Equation (7.32) can be proved by considering an arbitrary point $x \in P_I^{(k,2)}$, which is assumed to be non-empty. Substituting this point in the minimum equality system of $P_I^{(k,2)}$ and summing over all equations, weighted by the corresponding scalars λ, results in equation (7.32). ■

As in the case of $P_I^{(k,1)}$ (Propositions (7.10) and (7.11)), it can be shown that, for $n \geq \max\{5, k-1\}$ and $P_I^{(k,2)} \neq \emptyset$, the inequalities $x_c \geq 0$ define facets of $P_I^{(k,2)}$, whereas inequalities $x_c \leq 1$ are redundant. The proofs are analogous. Evidently, constraints (7.1) define improper faces of $P_I^{(k,2)}$.

7.8 Constraint Programming formulations

This section examines the application of Constraint Programming (CP) methods for modelling multidimensional assignment problems. It also proposes algorithms integrating CP and IP for solving the $(k,s)AP_n$, by generalising the approach exhibited in Chapter 6.

It is expected that CP models will be more compact. It is also natural to exclusively incorporate the *all_different* predicate. Starting from the simplest case of $(2,1)AP_n$, it is easy to see that the CP formulation is:

$$\begin{aligned} & \textit{all\_different}(X_i : i = 0, ..., n-1), \\ & D_X = \{0, ..., n-1\} \end{aligned} \tag{7.44}$$

Hence, the above formulation provides n pairs (i, X_i), $i = 0, ..., n-1$. In other words, $x_{ij} = 1$ if and only if $X_i = j$ (see also [53], [89]).

Concerning the $(3,1)AP_n$, recall that a solution to this problem is a set of n disjoint triples. Hence, define variables X_i^1, X_i^2 and the auxilliary variable Z_i. We wish to form n triples (i, X_i^1, X_i^2), for $i = 0, ..., n-1$. Variables X_i^1 must be pairwise different and the same must hold for variables X_i^2. The formulation is the following:

$$\begin{aligned}
&\textit{all\_different}(X_i^1 : i = 0, ..., n-1)\\
&\textit{all\_different}(X_i^2 : i = 0, ..., n-1)\\
&D_X = \{0, ..., n-1\}
\end{aligned}$$

Notice again that $x_{ijk} = 1$ if and only if $X_i^1 = j$ and $X_i^2 = k$.

Generalising this approach, the formulation of $(k,1)AP_n$ is the following:

$$\begin{aligned}
&\textit{all\_different}(X_i^p : i = 0, ..., n-1), \text{for } \ p = 1, ..., k-1\\
&D_X = \{0, ..., n-1\}
\end{aligned}$$

This is a formulation involving $(k-1) \cdot n$ variables and $(k-1)$ *all_different* constraints. The variable $x_{m^1 \cdots m^k}$ of the IP model is 1 if and only if $X^1_{m^1} = m^2$, ..., $X^{k-1}_{m^1} = m^k$.

The constraints for the Latin square problem ($(3,2)AP_n$) and for the OLS problem ($(4,2)AP_n$) have already been given in Section 5.3.1, while the CP formulation for a triple of MOLS ($(5,2)AP_n$) was presented in Section 6.5.1. We examine the generalisation of this formulation to the problem of finding $k-2$ MOLS ($(k,2)AP_n$), where $k \leq n+1$ (Proposition 7.8). The formulation for the 3-MOLS problem was presented in Section 6.5.1. Let variables $X_{ij}^p, p = 1, ..., k-2$ denote the contents of the p^{th} Latin square. Let also variables Z_{ij}^{qr}, $q, r = 1, ..., n, q < r$, enforce the orthogonality between any of the $\binom{n}{2}$ pairs (q, r) of Latin squares, by imposing that $Z_{ij}^{rq} = X_{ij}^r + n \cdot X_{ij}^q$. Variables X_{ij}^p and Z_{ij}^{rq} have domains of cardinality n and n^2, respectively. *All_different* constraints are required to ensure that *(a)* cells in each row and column of all squares contain pairwise different values and *(b)* all ordered pairs of values appear exactly once in any pair of Latin squares (i.e. all Z_{ij}^{qr} variables are pairwise different, for a certain pair (L_q, L_r) of Latin squares. The CP formulation is the following:

$$\begin{aligned}
&\textit{all\_different}(X_{ij}^p : i = 0, ..., n-1), \text{ for } j = 0, ..., n-1, \ \ p = 1, ..., k-2\\
&\textit{all\_different}(X_{ij}^p : j = 0, ..., n-1), \text{ for } i = 0, ..., n-1, \ \ p = 1, ..., k-2\\
&\textit{all\_different}(Z_{ij}^{rq} : i, j = 0, ..., n-1), \text{ for } q, r = 1, .., k-s, \ q < r \qquad (7.45)\\
&Z_{ij}^{rq} = X_{ij}^r + n \cdot X_{ij}^q, \text{ for } i, j, = 0, ..., n-1, \ q, r = 1, .., k-2, q < r\\
&D_X = \{0, ..., n-1\}, \quad D_Z = \{0, ..., n^2 - 1\}
\end{aligned}$$

This is a model involving $(k-1) \cdot n^2$ variables and $\binom{k}{2}(1+n^2)$ $+2 \cdot (k-2) \cdot n$ constraints. Notice that the number of indices for each variable is 2, i.e. equal to parameter s, and that the number of X variables is $k-2$, i.e. $k-s$.

Hence, for $s \geq 2$, $k - s < n - 1$ (Proposition 7.8), the CP formulation requires $(k-s)$ n-ary variables with s indices, i.e. variables $X^p_{m^1 \cdots m^s}$, $p = 1, ..., k-s$, and $m^i \in \{0, ..., n-1\}$ for $i = 1, ..., p$. It also requires $\binom{n}{s}$ auxilliary variables again with s indices and domains of cardinality n^s, in order to enforce the "hyper-orthogonality" constraints among the X variables. These are variables $Z^{p^1 \cdots p^s}_{m^1 \cdots m^s}$, where $p^1 < ... < p^s$ and $p^i \in \{1, ..., k-s\}$ for $i = 1, ..., p$. The CP formulation for $(k,s)AP_n$ is the following:

$$\begin{aligned}
&\mathit{all\_different}(X^p_{m^1 \cdots m^s} : m^1 = 0, ..., n-1), \text{ for } m^2, ..., m^s = 0, ..., n-1, \quad p = 1, ..., k-s \\
&\vdots \\
&\mathit{all\_different}(X^p_{m^1 \cdots m^s} : m^s = 0, ..., n-1), \text{ for } m^1, ..., m^{s-1} = 0, ..., n-1, \quad p = 1, ..., k-s \\
&\mathit{all\_different}(Z^{p^1 \cdots p^s}_{m^1 \cdots m^s} : m^1, ...m^s = 0, ..., n-1), \text{ for } p^1, ..., p^s = 1, .., k-s, \; p^1 < ... < p^s (7.46) \\
&Z^{p^1 \cdots p^s}_{m^1 \cdots m^s} = \sum_{i=1}^{s} n^{i-1} \cdot X^{p^i}_{m^1 \cdots m^s}, \text{ for } m^1, ..., m^{s-1} = 0, ..., n-1, \quad p^1 < ... < p^s \\
&D_X = \{0, ..., n-1\}, \quad D_Z = \{0, ..., n^s - 1\}
\end{aligned}$$

These formulations can be used to solve the $(k,s)AP_n$ via CP methods. The central procedure to be applied is the algorithm of [77], which achieves generalised arc-consistency for *all_different* constraints. Recall that this algorithm runs in $O(p^2d^2)$ steps for p variables with domains of cardinality at most d. For an *all_different* constraint on the Z variables in (7.46), this algorithm requires $O((n^s)^2 \cdot (n^s)^2) = O(n^{4s})$ steps.

Algorithms integrating CP and IP are easy to design. First, recall algorithm *CPI* of Chapter 6, i.e. an algorithm incorporating IP within a CP search tree. This scheme is naturally generalised for the $(k,s)AP_n$. The main principle is to assign values to subsets of variables with CP and periodically call IP in order to *(a)* derive a tighter bound on the objective function or *(b)* prove infeasibility or *(c)* extend a partial CP assignment to a complete one.

For an extension of algorithm *IPC* (Chapter 6), one would have to incorporate CP techniques to preprocess each subproblem of the IP search tree. Observe that each constraint set of $(k,s)AP_n$ can be associated to an *all_different* constraint. As an example, consider one of the constraint sets of the OLS problem:

$$\sum \{x_{m^1m^2m^3m^4} : m^1, m^2 = 0, ..., n-1\} = 1, \text{ for } m^3, m^4 = 0, ..., n-1$$

The *all_different* constraint associated to this constraint set is

$$\begin{aligned}
&\mathit{all\_different}\{W_{m^1m^2} : m^1, m^2 = 0, ..., n-1\} \\
&D_W = \{0, ..., n^2 - 1\}
\end{aligned}$$

where each value of D_W has an $1-1$ correspondence to each ordered pair (m^3, m^4). Hence, two indices are retained explicitly in the CP variables, while the remaining two indices are incorporated in the domain. If constraint propagation implies that $(m^3, m^4) \notin D_{W_{m^1m^2}}$ then variable $x_{m^1m^2m^3m^4}$ of the IP model must be set to 0. Note that, by symmetry, an equivalent constraint would be:

$$\begin{aligned}
&\mathit{all\_different}\{W_{m^3m^4} : m^3, m^4 = 0, ..., n-1\} \\
&D_W = \{0, ..., n^2 - 1\}
\end{aligned}$$

where each value of D_W would have an $1-1$ correspondence to each ordered pair (m^1, m^2).

For the $(k,s)AP_n$ consider, without loss of generality, the constraint set:

$$\begin{aligned}\sum\{x_{m^1m^2\ldots m^k} \ :\ & m^1 \in M_1, \ldots, m^{k-s-1} \in M^{k-s-1}, m^{k-s} \in M_{k-s}\} = 1\\ & \text{for all } m^{k-s+1} \in M_{k-s+1}, m^{k-s+2} \in M_{k-s+2}, \ldots, m^k \in M_k\end{aligned}$$

There are s indices left "free", i.e. outside the sum, and $k-s$ used for the summation. An associated *all_different* constraint should incorporate all indices either in the variables or in their domains, exactly as for the OLS problem. The constraint is as follows:

$$\begin{aligned}&\textit{all\_different}\{W_{m^1\ldots m^{k-s}} : m^1, \ldots, m^{k-s} = 0, \ldots, n-1\} \\ &D_W = \{0, \ldots, n^s - 1\}\end{aligned} \qquad (7.47)$$

if $k - s \leq s$, i.e. $s \geq \frac{k}{2}$, and

$$\begin{aligned}&\textit{all\_different}\{W_{m^{k-s+1}\ldots m^k} : m^{k-s+1}, \ldots, m^k = 0, \ldots, n-1\} \\ &D_W = \{0, \ldots, n^{k-s} - 1\}\end{aligned} \qquad (7.48)$$

if $s \leq \frac{k}{2}$. The reason we have to distinguish between these two cases is that an *all_different* is feasible only if the number of variables is less than or equal to the cardinality of the domains (Remark 5.2 . Notice for example that, for $s < \frac{k}{2}$, constraint (7.47) is infeasible.

7.9 Concluding remarks

This chapter introduced IP and CP models for general assignment problems. The IP model establishes a framework for unifying the polyhedral analysis of all assignment polytopes belonging to this class. In particular, the dimension of the linear relaxation of all members of this class is derived. A hierarchy among assignment polytopes is naturally imposed. Focusing on the classes of axial and planar assignment polytopes, it has been proved that their dimension equals a sum of terms from Newton's polynomial. The potential of this unified approach is demonstrated by identifying a family of clique facets for all axial assignment polytopes, for $k \geq 3$. We also proposed CP models, which can be linked to the IP ones within a framework integrating CP and IP.

List of Figures

List of Tables

Bibliography

[1] Appa G., Mathematical programming formulations of the orthogonal Latin square problem. *LSEOR Working Paper Series:* LSEOR 01.37.

[2] Appa G, Magos D., Mourtos I., Janssen J.C.M. (2001): On the Orthogonal Latin Squares polytope. *Discrete Mathematics*, accepted subject to revision (URL: http://www.cdam.lse.ac.uk/Reports/Files/cdam-2001-04.pdf)

[3] Appa G., Magos D., Mourtos I. (2002): On the assignment polytope. Submitted to *Mathematical Programming.* (URL: http://www.cdam.lse.ac.uk/Reports/Files/cdam-2002-01.pdf)

[4] Appa G., Mourtos I., Magos D. (2002): Integrating Constraint and Integer Programming for the Orthogonal Latin Squares Problem. In van Hentenryck P. (ed.), Principles and Practice of Constraint Programming (CP2002), *Lecture Notes in Computer Science* **2470**, Springer-Verlag, 17-32.

[5] Applegate D., Bixby B., Chvátal V., Cook W. (2001): TSP cuts which do not conform to the template paradigm. In Jünger M., Naddef D. (eds.) *Computational Combinatorial Optimization, Lecture Notes in Computer Science* **2241**, Springer Verlag, 261-304.

[6] Atamtürk A., Nemhauser G.L., Savelsbergh M.W.P. (2000): Conflict graphs in solving integer programming problems. *European Journal of Operational Research* **121**, 40-55.

[7] Balas E. (1965): An additive algorithm for solving linear programs with zero-one variables. *Operations Research* **13**, 517-546.

[8] Balas E., Padberg M. (1976): Set partitioning: a survey. *SIAM Rev.* **18**, 710-760.

[9] Balas E., Saltzman M.J. (1989): Facets of the three-index assignment polytope. *Discrete Applied Mathematics* **23**, 201-229.

[10] Balas E., Qi L. (1993): Linear-time separation algorithms for the three-index assignment polytope. *Discrete Applied Mathematics* **43**, 201-209.

[11] Balas E. and Saltzman M.J. (1991): An algorithm for the three-index assignment problem. *Operations Research* **39**, 150-161.

[12] Bandelt H.J., Crama Y. and Spieksma C.R. (1994): Approximation algorithms for multi-dimensional assignment problems with decomposable costs. *Discrete Applied Mathematics* **49**, 25-50.

[13] Beale E.M.L., Forrest J.J.H. (1976): Global optimization using special ordered sets. *Mathematical Programming* **10**, 52-69.

[14] Bessiere C., Freuder E.C., Regin J.C. (1999): Using constraint meta-knowledge to reduce arc-consistency computation. *Artificial Intelligence* **107**, 125-148.

[15] Blair C., Jeroslow R.G., Lowe J.K. (1988): Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research* **13**, 633-645.

[16] Bockmayr A., Casper T. (1998): Branch and infer: a unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing* **10**, 187-200.

[17] Bollobás B. (2002): *Modern Graph Theory.* Springer

[18] Borndörfer R. (1998): Aspects of Set Packing, Partitioning and Covering. PhD thesis, *TU Berlin.*

[19] Bose R.C., Shrikhande S.S. (1960): On the construction of sets of mutually orthogonal Latin squares and the falsity of a conjecture of Euler. *Transactions of the American Mathematics Society* **93**, 191-209.

[20] Bose R.C., Shrikhande S.S., Parker T.E. (1960): Further results on the construction of sets of mutually orthogonal Latin squares and the falsity of Euler's conjecture. *Canadian Journal of Mathematics* **12**, 189-203.

[21] Burkard R.E., Rudolf R., Woeginger G.J. (1996): Three-dimensional assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics* **65**, 123-139.

[22] Burkard R.E., Çela E. (1999): Linear Assignment problems and Extentions. In Pardalos P., Du D.Z. (eds.), *Handbook of Combinatorial Optimization,* Kluwer Academic Publishers.

[23] Chandru V., Hooker J.N. (1999): *Optimization methods for logical inference.* J.Wiley (NY).

[24] Christof T., Loebel A. (1997), The PORTA software, University of Heidelberg.

[25] Chvátal V. (1973): Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics* **4**, 305-337

[26] Chvátal V. (1975): On certain polytopes associated with graphs. *Journal of Combinatorial Theory Series B* **13**, 138-154.

[27] Cook W., Coullard C.R., Turán G. (1975): On the complexity of cutting plane proofs. *Discrete Applied Mathematics* **18**, 25-38.

[28] Cooper M.C. (1989): An optimal k-consistency algorithm. *Artificial Intelligence* **41**, 89-95

[29] Dantzig G.B. (1963): *Linear programming and extensions.* Princeton University Press.

[30] Darby-Dowman K., Little J., Mitra G., Zaffalon M. (1997): Constrain logic programming and integer programming approaches and their collaboration in solving an assignment scheduling problem. *Constraints* **1**, 245-264.

[31] Dash Associates (2001): *XPRESS-MP Optimiser Subroutine Library XOSL Version 12,* Reference Manual.

[32] Davis M., Putnam H. (1960): A computing procedure for quantification theory. *Journal of the ACM* **7**, 201-215.

[33] Dechter R. (1990): Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition. *Artificial Intelligence* **41**, 273-312.

[34] Dénes J., Keedwell A.D. (1974): *Latin Squares and their applications.* Academic Press, North Holland.

[35] Dénes J., Keedwell A.D. (1991): *Latin Squares: New developments in the Theory and Applications.* North-Holland.

[36] Euler L. (1849): Recherches sur une nouvelle espece de quarres magiques, Memoire de la Societe de Flessingue. *Commentationes arithmetice collectae* **2** (eloge St. Petersburg 1783), 302-361.

[37] Euler R., Burkard R.E., Grommes R. (1986): On latin squares and the facial structure of related polytopes. *Discrete Mathematics* **62**, 155-181.

[38] Euler R. (1987): Odd cycles and a class of facets of the axial 3-index assignment polytope. *Zastosowania Matematyki* **XIX**(3-4), 375-386.

[39] Euler R., Le Verge H. (1996): Time-tables, polyhedra and the greedy algorithm. *Discrete Applied Mathematics* **65**, 207-221.

[40] Freuder E.C., Wallace R.J., editors, (1998): Constraint Programming and Large Scale Discrete Optimization. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **57**, American Mathematics Society.

[41] Frieze A.M. (1974): A Bilinear programming formulation of the 3-dimensional assignment problem. *Mathematical Programming* **7**, 376-379.

[42] Frieze A.M., Yadegar J. (1981): An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *Journal of the Operational Research Society* **32**, 989-995.

[43] Frieze A.M. (1983): Complexity of a 3-dimensional assignment problem. *Journal of the Operational Research Society* **13**, 161-164.

[44] Gomes C., Shmoys D. (2002): The Promise of LP to Boost CSP Techniques for Combinatorial Problems. *CP-AI-OR'02*, 291-305, Le Croisic, France.

[45] Gomory R.E. (1960): Faces of an Integer Polyhedron. *Proceedings of the National Academy of Science* **57**, 16-18

[46] Grossmann I.E., Hooker J.N., Raman R., Yan H. (1997): Logic cuts for processing networks with fixed charges. *Computers and Operations Research* **21**, 265-279.

[47] Grötschel M., Padberg M.W. (1985): Polyhedral Theory. In Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B. (eds.) *The Traveling Salesman Problem: a guided tour of combinatorial optimization,* 251-305, J.Wiley & Sons.

[48] Haley K.B. (1963): The multi-index problem. *Operations Research* **11**, 368-379.

[49] Van Hentenryck P., Deville Y., Teng C.M. (1992): A generic arc-consistency algorithm and its specialisations. *Artificial Intelligence* **57**, 291-321.

[50] Hoffman K.L., Padberg M. (1993): Solving airline crew scheduling problems by branch-and-cut. *Management Science* **39**, 657-682.

[51] Hooker J.N. (1992): Generalised resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence* **6**, 271-286.

[52] Hooker J.N., Osorio M.A. (1999): Mixed logical/linear programming. *Discrete Applied Mathematics* **96-97**, 395-442.

[53] Hooker J.N. (2000): *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* J.Wiley(NY).

[54] Jeroslow R.E. (1987): Representability in mixed integer programming, I: Characterization results. *Discrete Applied Mathematics* **17**, 223-243.

[55] Karp R.M. (1972): Reducibility among combinatorial problems. In Miller R.E., Thatcher J.W. (eds.), *Complexity of Computer Computations*, 85-103, Plenum Press, NY.

[56] Kondrak G., van Beek P. (1997): A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence* **89**, 365-387.

[57] Kumar V. (1992): Algorithms for Constraint Satisfaction Problems: a Survey. *Artificial Intelligence* **13**, 32-44.

[58] Lam C.W.H. (1991): The Search for a Finite Projective Plane of Order 10. *Amer. Math. Monthly* **98**, 305-318.

[59] Land A.H., Doig A.G. (1960): An automatic method for solving discrete programming problems. *Econometrica* **28**, 497-520.

[60] Laywine C.F., Mullen G.L. (1998), *Discrete Mathematics using latin squares.* J.Wiley & Sons.

[61] Little J.D.C., Murty K.G., Sweeney D.W., Karel C. (1963): An algorithm for the traveling salesman problem. *Operations Research* **11**, 972-989.

[62] Magos D. (1993): *Solution Methods for Three-Dimensional Assignment Problems.* Ph. D. Thesis, Athens School of Economics and Business.

[63] Magos D., Miliotis P. (1994): An algorithm for the planar three-index assignment problem. *European Journal of Operational Research* **77** 141-153.

[64] Martello S., Toth P. (1987): Linear assignment problems. *Annals of Discrete Mathematics* **31**, 259-282.

[65] Milano M., van Hoeve W.J. (2002): Reduced cost-based ranking for generating promising subproblems. In van Hentenryck P. (ed.), Principles and Practice of Constraint Programming (CP2002), *Lecture Notes in Computer Science* **2470**, Springer-Verlag, 1-16.

[66] Miliotis P. (1978): Using cutting planes to solve the symmetric travelling salesman problem. *Mathematical Programming* **15**, 177-188.

[67] Mohr R., Henderson T.C. (1986): Arc and path consistency revisited. *Artificial Intelligence* **28**, 225-233.

[68] Nemhauser G.L., Wolsey L.A. (1988): *Integer and Combinatorial Optimization*, J.Wiley.

[69] Padberg M.W. (1973): On the facial structure of set packing polyhedra. *Mathematical Programming* **5**,199-215.

[70] Pierskalla W.P. (1967): The tri-substitution method for the three-dimensional assignment problem. *CORS Journal* **5**, 71-81.

[71] Pierskalla W.P. (1968): The multidimensional assignment problem. *Operations Research* **16**, 422-431.

[72] Poore A.B. (1994): Multidimensional assignment formulation of data association problems arising from multitarget and multisensor tracking. *Computational Optimisation and Applications* **3**, 27-57.

[73] Poore A.B., Robertson A.J. (1997): A new Lagrangian relaxation based algorithm for a class of multidimensional assignment problems. *Computational Optimisation and Applications* **8**, 129-150.

[74] Pulleyblank W.R. (1989): Polyhedral Combinatorics. In Nemhauser G.L., Rinnooy Kan A.H.G., Todd M.J. (eds.) *Optimization*, 371-446, North Holland.

[75] Pusztaszeri J.F., Rensing P.E., Liebling T.M. (1996): Tracking elementary particles near their primary vertex: a combinatorial approach. *Journal of Global Optimization* **9**, 41-64.

[76] Qi L., Balas E., Gwan G. (1993): A new class of facet-defining inequalities for the three index assignment polytope. In Dong-Zu Du and Jie Sun (eds.) *Advances in Optimization and Approximation*, 256-274, Kluwer Academic Publishers.

[77] Regin J.C. (1994), A filtering algorithm for constraints of difference in CSPs. *Proceedings of National Conference on Artificial Intelligence*, 362-367.

[78] Savelsbergh M.W.P. (1994), Preprocessing and Probing for Mixed Integer Programming Problems. *ORSA Journal on Computing* **6**, 445-454.

[79] Spieksma F.C.R. (2000): Multi-index assignment problems: complexity, approximation, applications. In Pitsoulis L., Pardalos P. (eds.), *Nonlinear Assignment Problems, Algorithms and Applications*, 1-12, Kluwer Academic Publishers.

[80] Stinson D.R. (1984): A short proof of the non-existence of a pair of orthogonal Latin squares of order 6. *Journal of Combinatorial Theory, series A* **36**, 373-376.

[81] Tarry G. (1900): Le Problème de 36 officiers. *C. R. Assoc. France Av. Sci.* **29**, Part 2, 170-203.

[82] Trotter L.E. (1975): A class of facet producing graphs for vertex packing polytopes. *Discrete Mathematics* **12**, 373–388.

[83] Tsang E. (1993): *Foundations of Constraint Satisfaction.* Academic Press (London).

[84] Tsang E. (1998): No more "Partial" and "Full Looking Ahead". *Artificial Intelligence* **98**, 351-361.

[85] Williams H.P. (1976): Fourier-Motzkin elimination extension to integer programming problems. *Journal of Combinatorial Theory* **21**, 118-123.

[86] Williams H.P. (1977): Logical problems and integer programming. *Bulletin of the Institute of Mathematics and its Implications* **13**, 18-20.

[87] Williams H.P. (1995): Logic applied to integer programming and integer programming applied to logic. *European Journal of Operations Research* **81**, 605-616.

[88] Williams H.P., Wilson, J.M. (1998): Connections between Integer Linear Programming and Constraint Logic Programming - An Overview and Introduction to the Cluster of Articles. *INFORMS Journal on Computing* **10** (3), 261-264.

[89] Williams H.P., Yang H. (2001): Representations of the all-different Predicate of Constraint Satisfaction in Integer Programming. *INFORMS Journal on Computing* **13**, 96-103.

[90] Wilson, J.M. (1990): Compact Normal Forms in Propositional Logic and Integer Programming Formulation. *Computers and Operations Research* **17** (3), 309-314 .

[91] Wright S.J. (1997): *Primal-dual interior-point methods.* Society for Industrial and Applicable Mathematics (SIAM).

[92] Yan H., Hooker J.N. (1999): Tight representation of logical constraints as cardinality rules. *Mathematical Programming* **85**, 363-377.

[93] Zhang H., Hsiang J. (1994): Solving open quasigroup problems by propositional reasoning. *Proceedings of International Computer Symposium*, Hsinchu, Taiwan.