# Integrating Constraint and Integer Programming for the Orthogonal Latin Squares Problem

Gautam Appa[1], Ioannis Mourtos[1], and Dimitris Magos[2]

[1] London School of Economics
London WC2A 2AE, UK
{g.appa,j.mourtos}@lse.ac.uk
[2] Technological Educational Institute of Athens
12210 Athens, Greece
dmagos@teiath.gr

**Abstract.** We consider the problem of Mutually Orthogonal Latin Squares and propose two algorithms which integrate Integer Programming (IP) and Constraint Programming (CP). Their behaviour is examined and compared to traditional CP and IP algorithms. The results assess the quality of inference achieved by the CP and IP, mainly in terms of early identification of infeasible subproblems. It is clearly illustrated that the integration of CP and IP is beneficial and that one hybrid algorithm exhibits the best performance as the problem size grows. An approach for reducing the search by excluding isomorphic cases is also presented.

## 1 Introduction and Definitions

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. Latin squares are multiplication tables of *quasigroups* ([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. This definition is extended to sets of $k > 2$ Latin squares, which are called *Mutually Orthogonal (MOLS)* if they are pairwise orthogonal. There can be at most $n-1$ MOLS of order $n$ ([10]).

A related concept is that of a *transversal*. A transversal of a Latin square is defined as a set of $n$ cells, each in a different row and column, which contain pairwise different values. As an example, consider the bordered cells of the second square in Table 1. It is easy to prove that a Latin square has an orthogonal mate if and only if it can be decomposed into $n$ disjoint transversals.

MOLS are closely related to finite algebra, in particular to theories of hypercubes, affine & projective planes and $(t, m, s)$-nets ([5]). Apart from their theoretical properties, they also possess interesting applications, mainly in multivariate statistical design and optimal error-correcting codes. Recently, they have been applied to problems related to tournament design and conflict-free access to parallel memories (see [10]).

P. Van Hentenryck (Ed.): CP 2002, LNCS 2470, pp. 17–32, 2002.

**Table 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 |

This work aims at identifying pairs of OLS for orders up to 12 and triples of MOLS of orders up to 10. Infeasible and unsolved problem instances are included. For example, it is well known that a pair of OLS of order 6 does not exist, whereas it remains unknown whether a triple of MOLS of order 10 exists. Both *Integer Programming* (IP) and *Constraint Programming* (CP) methods are applied. We report on the comparative performance of CP and IP on this feasibility problem, along with two algorithms that integrate both methods. This defines our broader aim, which is to investigate the potential of integrating CP and IP. Propositional reasoning has been successfully applied to solve open quasigroup problems ([17]), while recent work ([7]) has tested a CP/LP algorithm on the problem of quasigroup completion, commenting on the comparative superiority this scheme.

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. The necessity to solve large scale optimisation problems ([6]), together with the revelation of strong links between logic and optimisation ([3]), have stimulated a strong interest in successfully integrating IP and CP.

The combinatorial optimisation problems (COP) targeted by both methods are of the following generic form, hereafter called $COP(x, f, C, D)$:

$$min\{f(x) : x \in C, x \in D\} \tag{1}$$

In this formulation, $x$ is the vector of variables, $f$ the objective function, $D$ the external product of variable 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 (1), called $REL(x, f, C', D')$, is defined as:

$$min\{f(x) : x \in C', x \in D'\} \tag{2}$$

where $C' \subseteq C$ and $D' \supseteq D$, i.e. (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 (2) includes that of (1). Hence, a solution to (2) is not necessarily a solution to (1). If, however, (2) has no solution, neither does (1).

Every algorithmic method for solving (1) adopts further assumptions about the form of $f, C$ and $D$. IP requires that both the objective function $f$ and the set of constraints $C$ are linear. This fact highly restricts its declarative power, but

allows it to produce efficient problem relaxations. In CP, constraints can be of arbitrary type, although there is a broadening menu of specific constraint types, which are universally used in the CP literature. An example is the *all_different* predicate, which states that certain variables must be assigned pairwise different values (see [13, 15]).

The rest of this paper is organised as follows. Section 2 exhibits the IP and CP modes for the MOLS problem. Section 3 discusses a generic scheme for integrating CP and IP. An outline of all the algorithms is presented in Section 4. Computational results are discussed in Sections 5 and 6.

## 2 CP and IP Models for the MOLS Problem

Consider 4 $n$-sets $I, J, K, L$ and let $I$ be the row set, $J$ the column set and $K, L$ the sets of values for the two squares. Let the binary variable $x_{ijkl}$ be 1 if the pair of values $(k, l)$ appears in cell $(i, j)$ and 0 otherwise. Since each pair must occur exactly once, it follows that $\sum\{x_{ijkl} : i \in I, j \in J\} = 1$, for all $(k, l)$. Five more constraints of this type are formed by taking into account that the roles of the 4 sets are interchangeable. The result is the following IP model (also in [4]):

$$\sum\{x_{ijkl} : i \in I, j \in J\} = 1, \forall k \in K, l \in L \tag{3}$$

$$\sum\{x_{ijkl} : i \in I, k \in K\} = 1, \forall j \in J, l \in L \tag{4}$$

$$\sum\{x_{ijkl} : i \in I, l \in L\} = 1, \forall j \in J, k \in K \tag{5}$$

$$\sum\{x_{ijkl} : j \in J, k \in K\} = 1, \forall i \in I, l \in L \tag{6}$$

$$\sum\{x_{ijkl} : j \in J, l \in L\} = 1, \forall i \in I, k \in K \tag{7}$$

$$\sum\{x_{ijkl} : k \in K, l \in L\} = 1, \forall i \in I, j \in J \tag{8}$$

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

This is a $0-1$ IP model consisting of $6n^2$ constraints and $n^4$ binary variables, which also defines the *planar 4-index assignment problem* ($4PAP_n$) (in [1]).

The CP formulation of the OLS problem is easier to devise. Let the two squares be denoted as $X$, $Y$ and $X_{ij}$, $Y_{ij} \in \{0, ..., n-1\}$ be the variables denoting the value assigned to the cell $(i, j)$ in the two squares. For each square, an *all_different* predicate on the $n$ cells of every row and column ensures that the squares are Latin. To express the orthogonality condition we define the variables $Z_{ij} = X_{ij} + n \cdot Y_{ij}$, for $i, j = 0, 1, \ldots, n-1$. There are $n^2$ possible values for $Z_{ij}$, i.e. $Z_{ij} \in \{0, \ldots, n^2 - 1\}$, which have a $1-1$ correspondence with all $n^2$ ordered pairs $(i, j)$, for $i, j = 0, 1, \ldots, n-1$. The two squares are orthogonal iff all $Z_{ij}$s are pairwise different. The CP model for the OLS problem is exhibited below.

$$\textit{all\_different}\{X_{ij} : i \in I\}, \ \forall\ j \in J \tag{9}$$

$$\textit{all\_different}\{X_{ij} : j \in J\}, \ \forall\ i \in I \tag{10}$$

$$all\_different\{Y_{ij} : i \in I\},\ \forall\ j \in J \tag{11}$$
$$all\_different\{Y_{ij} : j \in J\},\ \forall\ i \in I \tag{12}$$
$$all\_different\{Z_{ij} : i \in I, j \in J\} \tag{13}$$
$$Z_{ij} = X_{ij} + n \cdot Y_{ij},\ \forall\ i \in I, j \in J \tag{14}$$
$$X_{ij}, Y_{ij} \in \{0, ..., n-1\},\ Z_{ij} \in \{0, ..., n^2 - 1\},\ \forall\ i \in I, j \in J$$

It is not difficult to establish the equivalence between the constraint sets of the two models. For example, (9) is the equivalent of constraint set (5). Clearly, the CP model is more compact, requiring $3n^2$ variables and $n^2 + 4n + 1$ constraints. The extensions of these models to the problem of identifying sets of $k$ MOLS ($k \leq n-1$) results in an IP model of $n^{k+2}$ variables and $\binom{k+2}{2} \cdot n^2$ constraints, whereas the CP model still requires only $O(n^2)$ variables and constraints. It follows that IP alone is impractical to handle large instances of the MOLS problem.

## 3 Integrating CP and IP

CP and IP models are solved using analogous algorithmic schemes. Both methods apply the "Divide & Conquer" paradigm: the initial problem is recursively divided into subproblems by partitioning the domain of at least one variable. A search tree is formed, where each node corresponds to a subproblem, the top node representing the initial problem. Both methods are *exact*, in the sense that they guarantee a complete search.

IP can be efficiently used to solve logic structures, as discussed in [3], while it is also possible to embed logic within a classical optimisation framework (see [9]). However, 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 prospect of integration will be discussed with respect to the next generic algorithm.

**Algorithm 1** *At each node/subproblem:*
*Preprocess* $COP(x, f, C, D)$; ***(I)***
***if*** *(feasible)*
***repeat***
**{**
*Solve* $REL(x, f, C', D')$; ***(II)***
*Infer additional constraints;* ***(III)***
**}**
***until*** $(x \in D)$ ***or*** *(infeasible)* ***or*** *(no inference)*
***if*** *(no inference)* ***and not***$((x \in D)$ ***or*** *(infeasible))*
*Create subproblems;* ***(IV)***
***return****;*

Concerning IP, step (I) involves preprocessing of variables ([14]). By using logical relations among the variables and information from the variables already

fixed, additional variable fixing is attempted. Step (II) implies solving the *LP-relaxation* of the problem, derived by dropping the integrality constraints. The advantage of this relaxation is that it provides an assignment of values to all variables and, if infeasible, it implies infeasibility for the original (sub)problem. If an LP-feasible but non-integer solution is produced, IP can proceed further by introducing additional inequalities, which cut-off this fractional solution and restrict further the feasible region $D'$ of the relaxation. These additional constraints, called *cutting planes*, are derived from the initial set of linear constraints $C$ by enforcing the fact that a subset of the variables must be integer. This process of solving the relaxation and extracting further inference in the form of cutting planes is repeated until the problem becomes infeasible or a feasible integer solution is found or no additional constraints can be generated. Identifying a cutting plane, which cuts off a certain fractional point, constitutes the *separation problem*. Although this can generally be a task as difficult as the original problem, i.e. they can both be $\mathcal{NP}$-complete, it is possible to separate certain classes of cutting planes in polynomial time (see [12]).

For CP, steps (I) & (III) are essentially parts of the same process: the domains of uninstantiated variables are reduced in order to avoid examining values that will lead to infeasible subproblems. The central notion in this approach is that of *k-consistency* (see [15] for definitions). 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, e.g. the *all_different* predicate, for which *full hyperarc-consistency* ([9]) can be accomplished in reasonable time (in [13]). Again, domain reduction may have further repercussions, therefore the process is repeated until no more domain values can be removed. CP lacks a proper relaxation for step (II), although recent work (in [9]) explores the potential of discrete relaxations. For both methods, Step (IV) is conducted by selecting an uninstantiated variable and splitting its domain to create two or more subproblems.

The foregoing discussion suggests that CP & IP can be integrated, at least in an algorithmic sense. For step (I), logical preprocessing conducted by IP is a subset of the sophisticated tools available for CP. IP can contribute by providing 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 discrete solution space of the original problem in contrast to IP, which reduces the continuous solution space of the relaxation (see also [2] and [8]). Finally, one of the two, or both, methods can be applied at step (IV) to determine the partition of the subproblem.

Note that 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 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.

## 4 Hybrid Algorithms

This section presents an outline of the algorithms implemented to solve the OLS problem. In total, five algorithms were examined, namely BB, BC, FC, IPC and CPI. The first three employ either IP or CP techniques. Algorithm *BB* is a simple *Branch & Bound* scheme. Algorithm *BC* is a *Branch & Cut* scheme and algorithm *FC* is a *Forward Checking* scheme, which implements various levels of constraint propagation.

The last two algorithms, namely *IPC* and *CPI*, integrate both methods. Note that, in accordance with Algorithm 1, the form of integration is basically determined by which method implements step (IV). This choice determines the form of the search tree. Hence, step (IV) can be implemented by:

**(A)** only CP, embedding IP within the CP search tree;
**(B)** only IP, embedding CP within the IP search tree;
**(C)** either IP or CP, switching between CP and IP search trees.

Algorithm CPI follows option (A), while algorithm IPC follows option (B).

We discuss the components of the CP and IP solver, i.e. the form of algorithms BC and FC, in 4.1 and 4.2, respectively. Preliminary variable fixing, which exploits problem symmetry, is illustrated in 4.3. In 4.4, we present the branching rule, which is common to all schemes for assessment purposes. Finally, the exact form of the algorithms is presented in 4.5.

### 4.1 CP Components

A node of the CP search tree is created whenever an uninstantiated variable is assigned a value still existing in its domain. The algorithm selects a certain cell $(i_0, j_0)$ such that $X_{i_0 j_0}$ is not instantiated. It then selects the smallest value $x \in D_{X_{i_0 j_0}}$ not already examined, sets $X_{i_0 j_0} = x$ and repeats this process for $Y_{i_0 j_0}$ if it is still uninstantiated. Thus it fixes a pair of cells, creating up to 2 successive nodes in the search tree. The orthogonality constraint is checked by maintaining auxiliary $0-1$ variables $C_{kl}$, in addition to variables $Z_{ij}$. Each such variable is initially 0 and is set to 1 whenever the pair $(k, l)$ is assigned to a particular pair of cells.

Concerning domain reduction, setting $X_{i_0 j_0} = k_0$ requires the deletion of value $k_0$ from the domains of all variables $\{X_{i_1 j_1} : i_1 = i_0 \text{ or } j_1 = j_0\}$. This reflects the fact that each value must appear exactly once in every row and column of each Latin square. The same procedure is applied when setting $Y_{i_0 j_0} = l_0$. Value $l_0$ is also removed from the domain of any $Y_{i_1 j_1}$ such that $X_{i_1 j_1} = k_0$. This enforces the orthogonality constraint.

In CP terms, this procedure achieves 2-consistency in $O(n)$ steps. If the cardinality of a domain becomes one, the variable is instantiated to this single

remaining value and the propagation routine is recursively called until no more variables can be fixed. If a domain is annihilated the node is declared infeasible, whereas if all variables are instantiated the algorithm terminates and returns the solution.

An additional level of consistency can be achieved by utilising the following lemma, presented here for the rows of square $X$.

**Lemma 1.** *Let* $S_{i_0} = \{X_{ij} : i = i_0$ *and* $X_{ij}$ *uninstantiated*$\}$ *and* $D_{i_0} = \{\bigcup D_{ij} : X_{ij} \in S_{i_0}\}$, $i_0 = 0, \ldots, n-1$. *A necessary condition for the existence of a feasible solution, at any subproblem, is that* $|S_{i_0}| \leq |D_{i_0}|$ *for each* $i_0 \in I$.

This lemma is valid for the variable set of an arbitrary *all_different* constraint. Computing the union of domains at each step, could pose a significant amount of work. For the problem in hand, the conditions of the above lemma are checked by introducing the notion of "degrees of freedom". For square $X$, the degrees of freedom for a certain row $i$ and value $k$, denoted by $XRDF_{ik}$, are the number of cells in this row, which still have value $k$ in their domains. The degrees of freedom for columns of $X$ ($XCDF_{jk}$) and for rows and columns of $Y$ are defined similarly. It is easy to see that $XRDF_{i_0k_0} = 0$ for some $k_0$, if and only if $|S_{i_0}| > |D_{i_0}|$ or $X_{i_0j} = k_0$ for some $j$. Consider the example of $D_{X_{11}} = D_{X_{12}} = D_{X_{13}} = \{1, 2\}$, where $X_{11}, X_{12}, X_{13}$ are the only uninstantiated variables of the row 1 of square $X$. Although no domain is empty, the problem is clearly infeasible. 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, such that $XRDF_{1k_0} = 0$. Degrees of freedom can be updated in $O(n)$ time in each node. Again, if $XRDF_{i_0k_0} = 1$, the single variable in row $i_0$, which has value $k_0$ in its domain, is assigned this value, no matter which other values are still in its domain.

As noted in Section 3, the particular structure exhibited by the *all_different* predicate allows for additional consistency checks to be performed. For example, let $D_{X_{11}} = D_{X_{12}} = \{1, 2\}$ and $D_{X_{13}} = D_{X_{14}} = \{1, 2, 3, 4\}$ be the only uninstantiated variables/cells of row 1. It is easy to see that values $1, 2$ must be deleted from $D_{X_{13}}, D_{X_{14}}$. Such cases are captured by the filtering algorithm presented in [13]. This algorithm runs in $O(p^2d^2)$ steps for a constraint on $p$ variables with domains of cardinality at most $d$. Hence, we need $O(n^4)$ steps for each of the constraints (9)-(12), i.e. $O(n^5)$ steps in total, and $O(n^8)$ steps for constraint (13). Being significantly more expensive, this filtering scheme is applied periodically, i.e. only after a certain number of variables have been fixed.

### 4.2 IP Components

The IP algorithm divides the initial problem recursively into subproblems by incorporating the concept of *Special Ordered Sets of type I (SOS-I)*. Observe that the IP model consists entirely of equalities. 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. Assume $S_1, S_2 \subset S$ such that $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$. Then the single variable of $S$ set to 1 will be

either in $S_1$ or in $S_2$. Thus the problem can be partitioned into two subproblems, each defined by setting the variables of either $S_1$ or $S_2$ to 0. The set $S$ is called an *SOS-I*. By recursively partitioning sets $S_1$ and $S_2$, eventually a subproblem will be left with a single variable of $S$ not set to 0, which is bound to be 1. If $|S_1| = |S_2| = \frac{n}{2}$, this occurs at most after $\lceil 2log_2 n \rceil$ partitions (levels of the tree), since each constraint involves $n^2$ variables. The search then proceeds by selecting another equality, whose left-hand side has at least one variable still not set to 0. It can be proved that, by branching on *SOS-I* instead of single $0-1$ variables, the depth of the search tree is reduced by a logarithmic factor (see also [11]).

Integer preprocessing is performed at each node in order to fix the values of additional variables before the LP is solved. Note that each variable appears in exactly 6 constraints. If $x_{i_0 j_0 k_0 l_0}$ is set to 1, all variables appearing in the same constraints as $x_{i_0 j_0 k_0 l_0}$ are set to 0, i.e. a total of $6(n-1)^2$ variables. Although this would also be enforced by the LP, implementing this "redundant" variable fixing can detect an infeasible node prior to solving the corresponding LP. It also accelerates the solution of the LP. Observe that the LP can also detect infeasibility arising from an empty domain or from a degree of freedom becoming 0. For example, if $D_{X_{i_0 j_0}} = \emptyset$ , the constraint $(i_0, j_0)$ of (8) is violated, while $XCDF_{j_0 k_0} = 0$ implies that the constraint $(j_0, k_0)$ of (5) is violated. The only types of infeasibility not detected by the LP are some cases captured exclusively by the filtering algorithm for the *all_different* constraints.

Polyhedral analysis has been successfully applied to the OLS problem and has provided families of strong cutting planes. The convex hull of all $0-1$ vectors satisfying (3)-(8) is the OLS polytope $P_I$ =conv$\{x \in \{0,1\}^{n^4} : Ax = e\}$. The LP-relaxation of $P_I$ is the polytope $P_L = \{x \in \mathbb{R}^{n^4} : Ax = e, 0 \leq x \leq 1\}$. An inequality satisfied by all points in $P_I$ is called *valid*. A valid inequality, which is not dominated by any other valid inequality, is called a *facet*. Thus, facets are the strongest possible inequalities in a polyhedral sense. Valid inequalities are not satisfied by all points of $P_L$. Therefore, their addition to the initial constraint set restricts further the feasible region of the LP-relaxation. Adding all known facets is impractical because it increases dramatically the size of the LP. In contrast, adding them "on demand", i.e. only when violated by the current (fractional) LP solution is more efficient. In [1], two non-trivial classes of facets induced by *cliques* are identified. The term "clique" stands for a *maximal* set of variables, at most one of which can be set to 1. Whether such an inequality is violated by the current LP solution can be determined in $O(n^4)$ steps, i.e. in time linear in the number of variables, which is the lowest possible. Two more classes of valid inequalities are also linearly separable: lifted 5-hole and lifted antiweb inequalities. An important thing to note is that a lifted antiweb inequality essentially states that a pair of OLS does not exist for $n = 2$ or, equivalently, any $2 \times 2$ subsquares of a set of OLS of order $n$ must contain at least 3 distinct values.

Violated cutting planes are added at the top node of the IP search tree and then every $\lceil (n-1)log_2 n \rceil$ levels, i.e. every time at least $\frac{(n-1)}{2}$ variables have been fixed to value 1. Violated cutting planes are repetitively added for up to

$\frac{n}{2}$ iterations. Within each iteration, clique inequalities are separated first and lifted 5-hole and antiweb inequalities are examined only if no violated cliques emerge. This happens because clique inequalities are, on average, cheaper to generate. The process terminates if (i) an integer solution is found or (ii) infeasibility is detected or (iii) the maximum number of iterations is reached. In case (iii), cutting planes, which are not satisfied as equalities by the last fractional solution, are deleted in order to reduce the matrix size. The remaining ones are retained at the descendant nodes.

### 4.3 Dealing with Symmetry

Algorithms searching for a feasible solution should reduce redundant search by excluding symmetrical subproblems. The set theoretical definition of isomorphy is applicable to Latin squares, viewed as multiplication tables of quasigroups. Hence, two Latin squares are *isomorphic* if one can be obtained from the other by permuting its rows, columns and elements. Extending this concept to OLS, we call two pairs of OLS *isomorphic* 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. According to our notation, this is equivalent to permuting the sets $I, J, K$ and $L$ respectively. The following analysis reduces the solution space of the original problem by proving that any subproblems not examined are isomorphic to a subproblem included in the reduced solution space.

Consider a random pair of OLS of order $n$, represented by point $u \in P_I$. The simplest isomorphism 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 [1], facilitates this process. Writing $u^1 = u(0 \longleftrightarrow 1)_I$ implies that point $u^1$ represents a new pair of OLS derived from $u$ by interchanging the roles of members 0 & 1 of set $I$. In general, we can write $u^1 = u(m_1 \longleftrightarrow m_2)_M$, where $M = I, J, K, L$. Interchanges can be applied sequentially.

The first observation is that, by properly permuting the elements of sets $K$ and $L$, we can have the cells of the first row of both squares containing the integers $0, ..., n-1$ in natural order. 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 $X$ is also in natural order. A pair of OLS of this form is called *standardised*. Fixing these $3n-1$ cells already reduces the problem size by a factor of $(n!)^2 \cdot (n-1)!$ ([10]).

Our approach allows for further reduction. Consider cell $Y_{10}$ and observe that $D_{Y_{10}} = \{2, ..., n-1\}$, i.e. $Y_{10} \neq 0, 1$, since $Y_{00} = 0$ and pair $(1, 1)$ already appears in position $(0, 1)$. Assume a pair of OLS having $Y_{10} = w$, where $w \in \{3, .., n-1\}$, and let $u \in P_I$ be the corresponding integer vector. Construct 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 $(1, 0)$, which reduces the solution space by an additional factor of $(n-2)$.

**Table 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} | |

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, $D_{Y_{i0}} = \{1, 3, 4, ..., i-1, i+1\}$ for $i \in \{3, ..., n-1\}$. 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 pair of OLS after this preliminary variable fixing, is depicted in Table 2.

### 4.4 Branching Rule

We present an efficient branching strategy used in both CP and IP. Recall first that CP creates subproblems by fixing a cell of the square $X$ and then fixing the same cell in square $Y$. 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). The alternative is to always select a cell in a different row and column. The first method fixes rows (columns) of the squares, therefore selecting each time a different value for a cell in the current row. The second method does not need to select a different value 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, according to the orthogonality constraint, set all cells of square $Y$ to pairwise different values. Hence, by definition, this method fixes transversals of square $Y$.

A common criterion for branching, is to select the next variable in the way that maximises the domain reduction achieved. If the search proceeds by fixing the remaining $n-1$ cells of the second row in both squares, $n-1$ different values will appear in each square. Setting $X_{1j_0} = k_0$ implies the deletion of value $k_0$ from the domains of all variables $X_{i_0 j_0}$ for $2 \leq i_0 \leq n-1$, except for $i_0 = k_0$ (having fixed $X_{k_0 0} = k_0$ implies that $k_0 \notin D_{k_0 j_0}$). Hence, $(n-3)$ domain members are removed for each value $k_0$ appearing in the second row, the only exception being value $k_0 = 1$ which results in removing $n-2$ domain members. In total, $(n-2)\cdot(n-3)+(n-2)$ domain values are removed. Similarly, $(n-2)^2+(n-3)$ domain values are removed from square $Y$. The total number of values removed from the domains of the uninstantiated variables/cells is $\alpha(n) = 2(n-2)^2 + (n-3)$.

Using an analogous argument, it can be proved that by fixing a transversal of square $Y$ along with fixing the corresponding cells of square $X$ with the same value 0, $(n-1)(n-2)$ domain values are removed from square $X$ and $2n(n-3)$ domain values are removed from square $Y$. The total number of domain values removed is $\beta(n) = (n-1)(n-2) + 2n(n-3)$. It turns out that $\alpha(n) \leq \beta(n)$ for $n \geq 2$. Therefore, fixing transversals is computationally more beneficial, a fact also supported by experimental results.

For IP to employ an equivalent branching scheme, it is sufficient to always select the *SOS-I* among the equalities of constraint set (7). Fixing, for example, value 0 in row 1 is implemented by recursively partitioning the variable set of equality $(1, 0)$ of (7). In the worst case, after branching on all $n^2$ *SOS-I* emanating from (7), a solution is bound to be constructed. Given this branching rule, the maximum depth of the search trees created by CP and IP is $O(n^2)$ and $O(n^2 log_2 n)$, respectively.

### 4.5 The Algorithms

Algorithm $BB$ involves no problem specific features and is considered in order to compare our results with those of a commercial IP solver, namely XPRESS-MP [16]. The IP solver uses its own cutting planes and all tuning parameters are set to their default values. No general purpose CP software has been used. Algorithm BC incorporates all the IP components and the branching rule described in the previous sections. Algorithm FC maintains 2-consistency and updates all "degrees of freedom" at each node. Each *all_different* constraint is made full hyperarc-consistent only whenever a transversal has been fixed. Again, this procedure is applied for up to $\frac{n}{2}$ iterations or until no more domain filtering is achievable.

Algorithm IPC embeds CP within IP by propagating on *all_different* constraints as an additional preprocessing step at each node. The technical details are as follows: observe first that the constraint set (3) is equivalent to the constraint *all_different*$\{W_{(i+n\cdot j)} : i \in I, j \in J\}$, where $D_{W_{i+n\cdot j}} = \{0, ..., n^2 - 1\}$. This is valid since each ordered pair $(k, l)$ is uniquely mapped to a number $(k + n \cdot l) \in \{0, ..., n^2 - 1\}$. An analogous constraint arises from each of the five constraint sets (4)-(8), i.e. in total, 6 *all_different* predicates are formed. Let $V$ denote the set of the $n^4$ $0-1$ variables. At a certain node of the IP search tree, let $F \subseteq V$ be the set of fixed variables. Define $W = \{W_{i+n\cdot j} : x_{ijkl} \in V \backslash F$ for some $k \in K, l \in L\}$ and $D_{W_{i+n\cdot j}} = \{(k+n\cdot l) : x_{ijkl} \in V \backslash F\}$. The preprocessing step for (3) is the application of the filtering algorithm of [13] to the predicate *all_different*$(W)$. If value $(k + n \cdot l)$ is removed from $D_{W_{i+n\cdot j}}$, $x_{ijkl}$ is set to 0.This procedure is then applied to constraint sets (4)-(8) for up to $\frac{n}{2}$ iterations. The objective of this preprocessing is to fix additional variables and to detect infeasibility without having to solve the LP.

The second hybrid scheme, embedding IP within CP, is Algorithm CPI. It is based on the FC algorithm, the additional step being to call the IP solver whenever a transversal has been fixed. The current status of the variables' domains in CP is passed to IP in the form of variable fixing. Obviously, $x_{ijkl} = 0$

iff $k \notin D_{X_{ij}}$ or $l \notin D_{Y_{ij}}$ and $x_{ijkl} = 1$ iff $X_{ij} = k$ and $Y_{ij} = l$. Note that, before calling the IP solver, all possible domain reduction by CP has been achieved. The reason is not only to avoid solving an infeasible IP but also to detect cases where IP can prove infeasibility although CP cannot. The IP solver deals with a single node, adding cutting planes for up to $n$ iterations. The initial LP is solved by the Primal Simplex algorithm, whereas all other iterations use the Dual Simplex algorithm. This is the standard approach because adding violated inequalities makes the solution of the previous iteration infeasible for the primal problem but feasible for the dual. Although an objective function is meaningless, a random objective function is introduced, in order to avoid degeneracy in the dual problem. No branching is performed by IP, the aim being to either extend a partial solution to a complete one or to prune a branch as infeasible.

Note that both CP and IP models are active in both hybrid algorithms, i.e. there has been no decomposition of the problem. The rationale is to achieve the best possible inference by both methods and also to compare their performance. The cost of maintaining both models is easily seen to be negligible.

## 5 Computational Experience

The callable libraries of XPRESS-MP have been used to codify the IP components ([16]). The code for the CP components has been written in Microsoft C++ environment and is mainly problem specific. All experiments were conducted on a PC under WinNT, with a PentiumIII processor at 866MHz and 256Mb of main memory. The experiments presented in this section concern the identification of a pair of OLS for orders $n = 3, ..., 12$. The number of variables in the IP model range from 81 ($n = 3$) to 20736 ($n = 12$), while the number of constraints ranges between 54 and 864. The CP model starts with 27 variables for $n = 3$ and ends up with 432 variables for $n = 12$ (see Section 2). Each algorithm returns a solution or proves that the problem is infeasible for the case of $n = 6$, where a complete search is required. For this reason, this instance is also particularly interesting for assessing the performance of the algorithms.

Table 3 illustrates two performance indicators: the number of nodes created during the search and the time in seconds taken to solve each problem. The number of nodes is illustrated in logarithmic scale. All schemes present the same general behaviour, with complexity exploding after $n = 9$. FC creates significantly more nodes than BB, which itself is much worse than BC. Algorithm IPC constantly creates fewer nodes that IP, although the difference is not significant. The striking difference appears between FC and CPI, with the hybrid algorithm creating considerably fewer subproblems. This indicates that CPI prunes infeasible branches much earlier by complementing CP's inference strength with that of IP. Note that CPI's performance is always between those of FC and BC.

In terms of time, the first comment is that BB is the slowest. Algorithm FC is the fastest for orders up to 9, providing the fastest enumeration of the whole solution space for $n = 6$. For larger values of $n$, however, it is outperformed by BC. Again IPC is slightly better than BC. It appears that the extra overhead of

**Table 3.** Performance indicators

| NODES (log) | | | | | |
|---|---|---|---|---|---|
| n | BB | BC | FC | IPC | CPI |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 2.32 | 0.00 | 7.50 | 0.00 | 5.58 |
| 6 | 14.88 | 12.39 | 18.13 | 12.33 | 14.65 |
| 7 | 14.36 | 8.13 | 20.58 | 7.87 | 19.02 |
| 8 | 10.08 | 5.58 | 11.36 | 5.39 | 10.37 |
| 9 | 18.24 | 15.01 | 28.27 | 18.22 | 22.45 |
| 10 | 22.44 | 19.72 | 32.88 | 19.70 | 26.95 |
| 11 | 27.20 | 22.37 | 35.16 | 22.20 | 29.85 |
| 12 | 29.32 | 23.17 | 39.49 | 23.09 | 32.78 |

| TIME (sec) | | | | | |
|---|---|---|---|---|---|
| n | BB | BC | FC | IPC | CPI |
| 3 | 3.2 | 3.2 | 0.0 | 3.2 | 0.00 |
| 4 | 5.8 | 3.7 | 0.0 | 3.7 | 0.00 |
| 5 | 12.4 | 9.6 | 0.0 | 9.6 | 0.57 |
| 6 | 1,843 | 553 | 43 | 532 | 376 |
| 7 | 7,689 | 3,325 | 523 | 2,937 | 854 |
| 8 | 294 | 252 | 2.7 | 231 | 112 |
| 9 | 32,341 | 17,854 | 12,745 | 15,826 | 14,439 |
| 10 | 37,065 | 20,561 | 20,935 | 19,306 | 17,842 |
| 11 | 46,254 | 24,812 | 29,423 | 23,538 | 21,651 |
| 12 | 59,348 | 31,642 | 37,161 | 29,213 | 26,983 |

**Table 4.** Percentage of nodes pruned

| n | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|
| INFI | 0.0 | 0.0 | 0.532 | 0.657 | 0.416 | 0.227 | 0.321 | 0.315 | 0.264 | 0.250 |
| INFC | 0.0 | 0.0 | 0.0 | 0.046 | 0.052 | 0.059 | 0.032 | 0.032 | 0.048 | 0.024 |

preprocessing at each node is counteracted by the smaller number of subproblems created. Once more, the more consistent and robust performance is exhibited by algorithm CPI, which lies between BC and FC up to $n = 9$ and outperforms both thereafter, i.e. as problem size grows.

Table 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 pruned by CP in algorithm IPC. Hence, indicator INFC is the percentage of infeasible nodes in the search tree of IPC which were detected by CP during preprocessing, without having to solve them. 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. These nodes correspond to subproblems made already consistent by CP alone. Thus 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. Hence, 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 significant improvement accomplished by incorporating IP within CP (algorithm CPI) in terms of nodes. CP is also useful within IP but to a smaller extent.

Another measurement of the inference strength is the early identification of infeasible branches. A representative criterion, independent of the actual tree form, is the number of transversals fixed before pruning a node. We report on

algorithms BC, FC, IPC, CPI only for the infeasible case of $n = 6$. Algorithms BC, 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 FC 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 FC creates the largest number of nodes. Algorithm CPI is faster than algorithms BC 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, BC and IPC keep on solving LPs and adding cutting planes in between without pruning any further branches.

These observations have obvious implications for more general COP. Given that a combinatorial 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 Branch & Cut tree is possible only for very small problem instances. For larger instances, CP can provide substantial improvement by enumerating all partial assignments of 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 transversal offers such a convenient criterion.

## 6 Triples of MOLS

We conclude by briefly presenting the algorithm implemented for triples of MOLS. Having already discussed the superiority of hybrid algorithms, we apply this approach to triples of MOLS. Since the IP model would require $n^5$ binary variables, we have to decompose the problem. Assume Latin squares $X, Y$ and $U$ of order $n$, where $X, Y$ are as in Table 2 and the first row of $U$ contains integers $0, ..., n-1$ in natural order.

CP fixes a certain number of $t(< n-1)$ values in square $U$, each in $n-1$ cells of different rows and columns, and performs the appropriate domain reduction. The IP model for the pair $X, Y$ of OLS is then used, after adding $2tn$ extra equality constraints to ensure orthogonality with the partially completed square $U$. The IP model is handled by algorithm IPC and returns an integer solution or proves infeasibility. If no integer solution is found, the CP solver backtracks and provides the next partial instantiation of square $U$ before the IP solver is called again. If an integer solution is returned, the CP solver instantiates the

**Table 5.** Time to identify triples of MOLS of order $n$

| n | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|
| TIME (sec) | 2.34 | 11.83 | n/a | 685 | 1457 | 13873 | * |

remaining cells of square $U$, under the additional constraints for orthogonality with squares $X, Y$. If it succeeds, a solution for the overall problem is found and the algorithm terminates. If not, the IP solver is called again and the search in the IP tree is resumed. Hence, the algorithm interchanges between the two solvers and retains both search trees active. Since both CP and IP control the creation of subproblems in different stages, this hybrid scheme implements option (C) (see Section 4). Table 5 illustrates preliminary results about the time (seconds) taken to identify a triple of MOLS of order $n$, using $t = 1$. Note that no triple of MOLS exists for $n = 6$, therefore this instance was not examined. For $n = 10$, the algorithm was interrupted after 40,000 seconds, having explored at least 0.00538% of the solution space.

## References


[1] Appa G., Magos D., Mourtos I., Janssen J. C. M.: On the Orthogonal Latin Squares polytope. Submitted to Discrete Mathematics (2001). (URL: http://www.cdam.lse.ac.uk/Reports/reports2001.html) 19, 24, 25

[2] Bockmayr A., Casper T.: Branch and infer: a unifying framework for integer and finite domain constraint programming. INFORMS Journal on Computing, **10** (1998) 187-200. 21

[3] Chandru V., Hooker J. N.: Optimization methods for logical inference. J.Wiley (1999). 18, 20

[4] Dantzig G. B.: Linear Programming and extensions. Princeton Univ. Press (1963). 19

[5] Dénes J., Keedwell A. D.: Latin squares and their applications. Acad. Press (1974). 17

[6] Freuder E. C., Wallace R. J. (ed.): Constraint Programming and Large Scale Discrete Optimization. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **57**, Amer. Math. Soc (1998). 18

[7] Gomes C., Shmoys D.: The Promise of LP to Boost CSP Techniques for Combinatorial Problems, CP-AI-OR'02, 291-305, Le Croisic, France (2002). 18

[8] Hooker J. N., Osorio M. A.: Mixed logical/linear programming. Discrete Applied Mathematics, **96-97** (1994) 395-442. 21

[9] Hooker J. N.: Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. J.Wiley (2000). 20, 21

[10] Laywine C. F., Mullen G. L.: Discrete Mathematics using Latin squares. J.Wiley (1998). 17, 25

[11] Magos D., Miliotis P.: An algorithm for the planar three-index assignment problem. European Journal of Operational Research **77** (1994) 141-153. 24

[12] Nemhauser G. L., Wolsey L. A.: Integer and Combinatorial Optimization. J.Wiley (1988). 21

[13] Regin J. C.: A filtering algorithm for constraints of difference in CSPs. Proceedings of National Conference on Artificial Intelligence (1994), 362-367. 19, 21, 23, 27
[14] Savelsbergh M. W. P.: Preprocessing and Probing for Mixed Integer Programming Problems. ORSA J. on Computing, **6** (1994) 445-454. 20
[15] Tsang E.: Foundations of Constraint Satisfaction, Acad. Press (1993). 19, 21
[16] Dash Associates: XPRESS-MP Version 12, Reference Manual (2001). 27, 28
[17] Zhang H., Hsiang J.: Solving open quasigroup problems by propositional reasoning, Proc. of International Computer Symposium, Hsinchu, Taiwan (1994). 18