Solving the MOLR and Social Golfers Problems

Warwick Harvey and Thorsten Winterer

IC-Parc, Imperial College London

Abstract. We present a range of techniques for tackling the problem of finding sets of Mutually Orthogonal Latin Rectangles (MOLR). In particular, we use a construction that allows us to search for solutions of a particular form with much reduced effort, and a seeding heuristic for the MOLR problem that allows a local search approach to find much better solutions than would be possible otherwise. Finally, we use the MOLR solutions found to construct solutions to the social golfer problem that improve the best known number of rounds for 43 instances, by as many as 10 rounds.

1 Introduction

In [4], Dotú and Van Hentenryck used a constructive seeding heuristic to significantly improve the quality of the results they found using local search for the social golfer problem. As they noted, for certain instances their heuristic corresponds to constructing a complete set of Mutually Orthogonal Latin Squares (MOLS), and results in an optimal solution to the social golfers problem for those instances without any search required. There is a well-known construction [2–II.2.20] for complete sets of MOLS that works for more instances than Dotú and Van Hentenryck's heuristic, again yielding optimal solutions for the corresponding instances of the social golfers problem.

There are other methods that can be used to construct solutions to the social golfer problem using sets of MOLS, due to Sharma and Das [18] and `mathtalk-ga` [13]. These approaches also work when given a set of Mutually Orthogonal Latin Rectangles (MOLR), allowing solutions to the social golfer problem to be constructed for many cases where a set of MOLS of sufficient size is not known to exist.

Sets of MOLR are also useful for constructing solutions to other problems, for example perfect hash families [20] and low-density parity check codes [3]. However, it seems that little work has been done on this problem. Franklin [6] gives a set of 3 MOLR of order 9×10, while Wanless [22] improves this to a set of 4. Mullen and Shiue [15] give a simple construction that generates some useful sets of MOLR, mostly when the number of rows is small.

In this paper we present constructions and solutions that significantly extend and improve on the currently known results for the MOLR problem, and then use these to construct improved solutions for 43 instances of the social golfer problem.

In Section 2 we present background material on the MOLS, MOLR and social golfer problems and their constructions. In Section 3 we present a new direct construction for certain MOLR instances, and another construction that allows us to find MOLR solutions by solving a much simpler problem. The local search algorithms we applied to the MOLR and reduced problems are described in Section 4, while the results obtained from all the methods used are presented in Section 5.

P. van Beek (Ed.): CP 2005, LNCS 3709, pp. 286–300, 2005.

2 Background

2.1 Latin Squares and Latin Rectangles

A *Latin Square* of order n is an $n \times n$ array where each entry in the array is taken from the set $\{0 \ldots n-1\}$ and for each row and column the elements of that row/column are all different. A *Latin Rectangle* of order $m \times n$ ($m \leq n$) is the obvious generalisation of a latin square to a non-square array: an $m \times n$ array where each entry in the array is taken from the set $\{0 \ldots n-1\}$ and for each row and column the elements of that row/column are all different. Clearly, a latin rectangle of order $n \times n$ is just a latin square of order n.

For a latin square or rectangle L, we denote the element in row i and column j by $L(i,j)$ ($i \in \{0 \ldots m-1\}, j \in \{0 \ldots n-1\}$). A square or rectangle L is then latin if it satisfies the constraints:

$$L(i,j) \in \{0 \ldots n-1\} \quad \forall i \in \{0 \ldots m-1\}, \forall j \in \{0 \ldots n-1\} \tag{1}$$

$$\text{alldifferent}(L(i,j) | j \in \{0 \ldots n-1\}) \quad \forall i \in \{0 \ldots m-1\} \tag{2}$$

$$\text{alldifferent}(L(i,j) | i \in \{0 \ldots m-1\}) \quad \forall j \in \{0 \ldots n-1\} \tag{3}$$

A set of *Mutually Orthogonal Latin Squares* (MOLS) is a set of latin squares such that for any pair of squares L_α and L_β from the set, the ordered pairs $(L_\alpha(i,j), L_\beta(i,j))$ must be distinct for all i and j:

$$\text{alldifferent}((L_\alpha(i,j), L_\beta(i,j)) | i \in \{0 \ldots m-1\}, j \in \{0 \ldots n-1\}) \quad \forall \alpha, \beta \in \{1 \ldots r\}, \alpha \neq \beta \tag{4}$$

A set of *Mutually Orthogonal Latin Rectangles* (MOLR) is the straightforward generalisation of MOLS from latin squares to latin rectangles: $(L_\alpha(i,j), L_\beta(i,j))$ must be distinct for all i and j for any pair of distinct rectangles L_α and L_β from the set. Clearly, a set of MOLR of order $n \times n$ is just a set of MOLS of order n.

Let $N(n)$ be the maximum number of squares possible in a set of MOLS of order n; let $N(m,n)$ be the maximum number of rectangles possible in a set of MOLR of order $m \times n$. Clearly, $N(n) \leq N(m,n)$ for any $m \leq n$ since a set of MOLS of order n can be turned into a set of MOLR of order $m \times n$ by removing a suitable number of rows from the bottom of each square. We also have that $N(m,n) \leq n-1$ for any m such that $1 < m \leq n$. A set of MOLS or MOLR containing $n-1$ elements is said to be *complete*.

For $n = p^e$ for some prime p, there is a well-known construction [2–II.2.20] that yields a complete set of MOLS. Let $\text{GF}(n)$ be the finite field of order n. For each $\alpha \in \text{GF}(n) \backslash \{0\}$, let $L_\alpha(i,j) = \alpha i + j$, where $i, j \in \text{GF}(n)$ and the algebra is performed in $\text{GF}(n)$. The set $\{L_\alpha | \alpha \in \text{GF}(n) \backslash \{0\}\}$ is then a set of $n-1$ MOLS of order n.

Note that the existence of a complete set of MOLS for these values of n means that the $m \times n$ MOLR existence problem is solved as well: we have $N(n) = N(m,n) = n-1$. For other (non prime power) values of n — other than 6 [1] — the MOLS (and hence MOLR) problem is still open; the best known lower bound on $N(n)$ is generally much smaller than n (see [2–Table II.2.72]). For these n it is usually the case that $N(m,n) > N(n)$ if $m < n$, but prior to the current work, little was known about the value of $N(m,n)$ for these cases, even for small values of n.

[1] $N(6) = 1$; constructing two MOLS of order 6 is Euler's 36 Officers Problem, a famous problem with no solution.

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17

Fig. 1. Sharma and Das's construction: the first round

2.2 The Social Golfer Problem

The *Social Golfer Problem* [9–Problem 10] involves trying to schedule w rounds of golf, where in each round the $g \times s$ players are arranged into g groups of size s such that no pair of players appear in a group together more than once. This problem has received significant attention from the constraint programming community recently, for example [1,4,5,12,16,17,19]. It is well-known that certain instances of the social golfer problem (when $w(s-1) = gs-1$) correspond to instances of the *Resolvable Balanced Incomplete Block Design* problem. Similarly, it is known that when $g = s$, the w-round social golfer problem corresponds to the problem of finding $w-2$ MOLS of order g. This correspondence was exploited in [4], where the authors used a heuristic construction to seed their local search; when $g = s$ was prime, their construction corresponded to the standard construction for a complete set of MOLS, and thus yielded an optimal solution to the golf problem without any search required. Clearly by exploiting the full power of the MOLS construction, one can also obtain optimal search-free solutions for the cases where $g = s$ is a prime power.

Sharma and Das's Construction. There are two other constructions of which we are aware that allow solutions to the social golfer problem to be constructed from a set of MOLS (in practice, MOLR). The first is that of Sharma and Das [18]. This construction uses a set of r MOLR of order $m \times n$ to construct a solution to the social golfer problem with $g = n$ and $s = m$: if s does not divide g a solution with $w = r+1$ rounds is obtained; if s does divide g an extra round may be obtained.

Write the golfers out in an $s \times g$ array G, as shown in Figure 1. One round of the golf schedule is obtained from taking each column as a group.

$$\{G(i,j) | i \in \{0 \ldots s-1\}\} \quad \forall j \in \{0 \ldots g-1\}$$

The next r rounds are obtained using r MOLR of order $s \times g$. Each latin rectangle L_α yields a round, with each value appearing in L_α corresponding to a group in the round: a group contains those players in G that have the same corresponding value in L_α.

$$\{G(i,j) | L_\alpha(i,j) = k, i \in \{0 \ldots s-1\}, j \in \{0 \ldots g-1\}\} \quad \forall k \in \{0 \ldots g-1\}$$

This is illustrated in Figure 2, where L_α has been superimposed on G; for example, group 0 corresponds to players 0, 7 and 16. Finally, if s divides g then a further round can be obtained by dividing each row up into groups, as illustrated in Figure 3. □

In the case where s divides g and $g \geq s^2$, Sharma and Das's construction can be extended. In this case, rather than just one extra round involving groups lying entirely

$$\begin{array}{cccccc} 0^{0} & 1^{1} & 2^{2} & 3^{3} & 4^{4} & 5^{5} \\ 6^{1} & 7^{0} & 8^{3} & 9^{2} & 10^{5} & 11^{4} \\ 12^{2} & 13^{3} & 14^{4} & 15^{5} & 16^{0} & 17^{1} \end{array}$$

Fig. 2. Sharma and Das's construction: another round

within a row of G, one can actually schedule a w'-round mini-tournament amongst the players in the row, where w' is the best known number of rounds for the golf problem with g/s groups of size s. Doing this simultaneously for all the rows means that a schedule for $r+1+w'$ rounds can be achieved.

`mathtalk-ga`'s Construction. The other construction is due to `mathtalk-ga` [13]. This uses a set of r MOLS of order n to construct a solution with $g = w = n$ and $s = r+1$; again, if s divides g an extra round is possible.

As with Sharma and Das's construction, write the golfers out in an $s \times g$ array G. Each latin square is associated with a row of G after the first. The rows of the latin squares correspond to the rounds of the golf schedule and the columns to the groups of the rounds. Each entry in a latin square thus indicates which element of the latin square's corresponding row of G appears in a given group of a given round. The j^{th} player in the first row of G always appears in group j in each round. The groups for round i are thus:

$$\{G(0,j)\} \cup \{G(\alpha, L_\alpha(i,j)) | \alpha \in \{1 \ldots r\}\} \quad \forall j \in \{0 \ldots g-1\}$$

Since each group contains one player from each row of G, if s divides g an extra round is possible by dividing the rows into groups, as with Sharma and Das's construction. □

This construction can be adapted to work with MOLR instead of MOLS. If the best known set of MOLS is of insufficient size for the desired golf group size, a larger set of MOLR may be used instead, at the expense of a reduced number of rounds. In general, a set of r MOLR of order $m \times n$ allows the construction of a golf solution with $g = n$, $s \leq r+1$ and $w = m$ (if s does not divide g) or $w = m+1$ (if it does).

As before, more rounds can be achieved if s divides g and $g \geq s^2$, by scheduling parallel mini-tournaments of g/s groups of s amongst the players in the rows of G. Using known MOLS results, this immediately yields solutions for previously unsolved instances (g-s-w) 12-3-16, 18-3-26 (closing this instance) and 20-4-25.

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17

Fig. 3. Sharma and Das's construction: an extra round

3 MOLR Constructions

3.1 Constructing MOLR Solutions Directly

In this section we present a generalisation of the classic construction for a complete set of MOLS of order $n = p^e$ that allows us to generate (incomplete) sets of MOLR for other values of n. Specifically, our construction allows us to generate $p^e - 1$ MOLR of order $p^e \times n'$ for $n' = \prod_{i=1}^{e} q_i$ where $q_i = p$ or $q_i \geq 2p-1$ for $i \in \{1 \dots e\}$.

As noted above, for the MOLS construction we have $L_\alpha(i,j) = \alpha i + j$, where $\alpha, i, j \in \mathrm{GF}(p^e)$ and $\alpha \neq 0$ and the algebra is performed in $\mathrm{GF}(p^e)$. Essentially, the elements in column 0 of each square are generated by the product αi, with the rest of the columns being generated by adding the column index. We leverage this for other values of n by taking the columns generated for $n = p^e$ and extending them to rectangles of size $p^e \times n'$ by performing the addition of the column index in a different group, carefully selected to preserve the orthogonality of the resulting rectangles. The key property is:

$$\text{alldifferent}(-L_\alpha(i_2,0) + L_\alpha(i_1,0) | \alpha \in \{1 \dots r\}) \quad \forall i_1, i_2 \in \{0 \dots m-1\}, i_1 \neq i_2 \quad (5)$$

where $r = p^e - 1$ is the number of rectangles and $m = p^e$ is the number of rows in each rectangle. This property holds when the evaluation is done in $\mathrm{GF}(p^e)$ (or the MOLS construction would not work), and must hold in our chosen group of order n'.

One of the standard interpretations of $\mathrm{GF}(p^e)$ is as polynomials of degree at most $e-1$ with coefficients being elements of the integers modulo p. Multiplication, addition, etc. are then done as for polynomials, with polynomials of degree e or more being reduced modulo an irreducible polynomial of degree e. Now consider the polynomials of degree at most $e-1$ where the coefficients of x^{i-1} are elements of the integers modulo q_i. These polynomials form a group G of order $n' = \prod_{i=1}^{e} q_i$ where the group operation is addition. For our construction, we take the '0' columns constructed by the MOLS construction, interpret them using the polynomial interpretation above, and map the coefficients so that $a \pmod p$ is mapped to $a \pmod{q_i}$ for $a \in \{0 \dots p-1\}$. Each of the n' columns required to form the rectangles of the desired size are then constructed by adding different elements of G using the group operation of G.

These rectangles are mutually orthogonal if either $q_i = p$ or $q_i \geq 2p-1$ for all q_i. It suffices to show that

$$\begin{gathered} -b + a \neq -d + c \pmod p \Rightarrow -b + a \neq -d + c \pmod{q_i} \\ \forall a, b, c, d \in \{0 \dots p-1\} \end{gathered} \quad (6)$$

as this means that the constraint (5) is maintained when we switch to generating the columns in the group G rather than in $\mathrm{GF}(p^e)$. If $q_i = p$ then (6) is clearly satisfied. Suppose $q_i \geq 2p-1$, and consider $-b+a$ and $-d+c$ using normal integer arithmetic. These differences must fall in the range $\{-p+1 \dots p-1\}$. Each of these differences is mapped to a different equivalence class modulo q_i if $q_i \geq 2p-1$, and hence (6) holds.

Using the above construction yields, for example:

3 MOLR of order 4×6 $\quad (p = 2, n' = 2 \cdot 3)$
4 MOLR of order 5×10 $\quad (p = 5, n' = 10)$

7 MOLR of order 8×12 $(p = 2, n' = 2 \cdot 2 \cdot 3)$
6 MOLR of order 7×14 $(p = 7, n' = 14)$
8 MOLR of order 9×15 $(p = 3, n' = 3 \cdot 5)$
8 MOLR of order 9×18 $(p = 3, n' = 3 \cdot 6)$
7 MOLR of order 8×20 $(p = 2, n' = 2 \cdot 2 \cdot 5)$

Moreover, if one relaxes the condition to also allow $q_i \in \{p+1 \ldots 2p-2\}$, then one can obtain near-solutions to the MOLR problem. These solutions can be used to seed the local search (á là [4]), often giving a starting point with fewer violations than had been achieved using local search alone, and allowing previously unsolved instances to be solved. Note also that sometimes a near-solution constructed in this way can be turned into a solution of a slightly smaller instance simply by dropping one or more rectangles and/or rows; a proper investigation of this is beyond the scope of this paper.

3.2 Constructing MOLR Solutions by Solving a Reduced Problem

Many MOLR constructions follow the same basic pattern:

$$L_\alpha(i,j) = \alpha i + j$$

For the MOLS construction the computation is done in GF(n); for Mullen and Shiue's construction [15] it is done in Z_n (the integers modulo n); for our construction the product is computed in GF(p^e) and the addition in a group of order n. The basic premise is the same: the entry in column j of a row is computed by adding j to the entry in column 0, and the entries in column 0 of each rectangle are constructed or selected in such a way that the resulting rectangles are latin and mutually orthogonal.

This leads to a more general way to construct MOLR: if one wishes to find r MOLR of order $m \times n$, search for a set of r "0" columns of height m such that each of these columns can be extended to a full latin rectangle by adding j to form column j of the rectangle, and such that the resulting rectangles are mutually orthogonal. We refer to this reduced problem as the *column problem*.

Note that the addition in the construction can be performed in any group of order n. This choice must be reflected in the constraints of the column problem — in the rest of this section all arithmetic and algebra is assumed to be performed in the selected (possibly non-commutative) group. Note that the choice of group affects which instances can be solved in this way; for example, a solution for 4 MOLR of order 5×6 can be constructed if one uses the integers modulo 6, but not if one uses the other group of order 6.

There are two constraints for the column problem. The first is (5), the second is:

$$\text{alldifferent}(L_\alpha(i,0) | i \in \{0 \ldots m-1\}) \quad \forall \alpha \in \{1 \ldots r\} \tag{7}$$

The rectangles are then constructed as follows:

$$L_\alpha(i,j) = L_\alpha(i,0) + j \quad \forall \alpha \in \{1 \ldots r\}, \forall i \in \{0 \ldots m-1\}, \forall j \in \{0 \ldots n-1\} \tag{8}$$

Note that (8) implies that the entries in a row are distinct:

$$\text{alldifferent}(L_\alpha(i,j) | j \in \{0 \ldots n-1\}), \forall \alpha \in \{1 \ldots r\}, \forall i \in \{0 \ldots m-1\}$$

and (7) with (8) implies that the entries in a column are distinct:

$$\text{alldifferent}(L_\alpha(i,j)|i \in \{0\ldots m-1\}), \forall \alpha \in \{1\ldots r\}, \forall j \in \{0\ldots n-1\}$$

That is, the rectangles are latin.

It remains to show that the rectangles are orthogonal. Consider two cells in L_α at positions (i_1, j_1) and (i_2, j_2) that have the same value:

$$\begin{aligned} L_\alpha(i_1,j_1) &= L_\alpha(i_2,j_2) \\ \text{i.e.} \quad L_\alpha(i_1,0)+j_1 &= L_\alpha(i_2,0)+j_2 \\ \text{i.e.} \quad -L_\alpha(i_2,0)+L_\alpha(i_1,0) &= j_2-j_1 \end{aligned} \tag{9}$$

Then for any other rectangle L_β, orthogonality requires that the values in the same positions must be different; that is, we need to show that:

$$L_\beta(i_1,j_1) \neq L_\beta(i_2,j_2)$$

From (5) we know that:

$$\begin{aligned} -L_\beta(i_2,0)+L_\beta(i_1,0) &\neq -L_\alpha(i_2,0)+L_\alpha(i_1,0) \\ \text{i.e.} \quad -L_\beta(i_2,0)+L_\beta(i_1,0) &\neq j_2-j_1 \qquad \text{(by (9))} \\ \text{i.e.} \quad L_\beta(i_1,0)+j_1 &\neq L_\beta(i_2,0)+j_2 \\ \text{i.e.} \quad L_\beta(i_1,j_1) &\neq L_\beta(i_2,j_2) \end{aligned}$$

as required.

Thus solving the column problem allows us to construct a solution to the corresponding MOLR problem. Of course, it only allows us to find MOLR solutions of this particular form; there may be solvable MOLR instances for which there are no solutions of this form. For example, it is known that there are 2 MOLS of order 10, yet a complete search of the corresponding column problem found no solutions (when using either of the two distinct groups of order 10 for the arithmetic).

4 Local Search for MOLR

In this section we present the local search approach we used to tackle the MOLR problem.

4.1 MOLR Model

The evaluation function used by local search algorithms for satisfaction problems is usually based on the number of violated constraints. In order to minimise the number of constraints that have to be checked for violation, we chose a model for the MOLR problem that observes as many constraints as possible implicitly.

For r MOLR of order $m \times n$, our model contains $r+1$ rectangles. The rectangles L_1 through L_r are initialised such that each row contains a permutation of $\{0\ldots n-1\}$, fulfilling constraint (2). If we restrict the move operator to exchange values only within a row, constraint (2) will always be observed during the search.

Rectangle L_0 is a control rectangle, which is initialised as

$$L_0(i,j) = j \quad \forall i \in \{0 \ldots m-1\}, \forall j \in \{0 \ldots n-1\}$$

The control rectangle ensures that any assignment σ observing constraint (4) for all pairs of rectangles (L_0, L_α), $1 \leq \alpha \leq r$ will also observe constraint (3), since

$$(L_0(i,j), L_\alpha(i,j))_\sigma = (j, L_\alpha(i,j))_\sigma$$

and in any solution that observes constraint (4)

$$i_1 \neq i_2 \Leftrightarrow (j, L_\alpha(i_1,j))_\sigma \neq (j, L_\alpha(i_2,j))_\sigma \Leftrightarrow L_\alpha(i_1,j)_\sigma \neq L_\alpha(i_2,j)_\sigma$$

Using this model and restricting the search moves to exchange values only within a row, any value assignment σ that does not violate constraint (4) will also observe the two other constraints. Therefore, we can use the evaluation function

$$f(\sigma) = \sum_{\alpha=0}^{r-1} \sum_{\beta=\alpha+1}^{r} \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} max(0, \#(\sigma, \alpha, \beta, x, y))$$

$$\text{with} \quad \#(\sigma, \alpha, \beta, x, y) = |\{(i,j) | (L_\alpha(i,j), L_\beta(i,j))_\sigma = (x,y)\}| - 1$$

the sum of violations of constraint (4) for all pairs of rectangles. An assignment σ is a solution for an MOLR instance iff $f(\sigma) = 0$.

4.2 Neighbourhood

Since we only allow the exchange of values within a row, we generally use only one type of move operator, swapping the values of two cells within a row:

$$\mu(\alpha, i, j_1, j_2) = [L_\alpha(i,j_1) \leftrightarrow L_\alpha(i,j_2)]$$

For each search step, the algorithm chooses the move that reduces the number of violations most. Should there be more than one move with the same benefit, a move will be randomly chosen from that list. Since a swap between two cells can only improve the violation count if at least one of the cells is currently in violation, we can restrict the neighbourhood to such moves. The neighbourhood can then be defined as:

$$\mathcal{S} = \{\mu(\alpha, i, j_1, j_2) | \exists \beta \in \{0 \ldots r\} : \beta \neq \alpha \wedge$$
$$(\#(\sigma, \alpha, \beta, L_\alpha(i,j_1), L_\beta(i,j_1)) > 1 \vee \#(\sigma, \alpha, \beta, L_\alpha(i,j_2), L_\beta(i,j_2)) > 1)\}$$

However, if there is no move that will improve the evaluation of the assignment, the algorithm will choose with probability p_1 a move that least increases the violation count; with probability p_2, it will swap two randomly chosen values within a randomly chosen row; and with probability p_3, it will right-shift a randomly chosen row by a random amount.

The right-shift move (shifting row i of L_α by w positions) can be defined as:

$$\bar{\mu}(\alpha, i, w) = [L_\alpha(i, (j+w) \bmod n) \leftarrow L_\alpha(i,j), \forall j \in \{0 \ldots n-1\}]$$

Right-shifting a complete row makes a larger step away from the local minimum, and often allows the algorithm to escape from the area of that minimum. In our experiments, we set $p_1 = 0.4$, $p_2 = 0.35$, and $p_3 = 0.25$. These values were chosen since they gave good results in our initial tests.

4.3 Tabu Search Algorithm

Our local search algorithm is based on a tabu search with restart. For each possible move $\mu(\alpha, i, j_1, j_2)$ or $\bar{\mu}(\alpha, i, w)$, the tabu table $\mathcal{T}$ contains the step number $t(\alpha, i, j_1, j_2)$ or $t(\alpha, i, w)$, respectively, until which this move is tabu. Moves that are currently tabu are filtered from the neighbourhood. The tabu is based on the location of the swapped cells, or the row and the shift, not on the particular values that are swapped: when the values of two cells are swapped, a swap between these two cells becomes tabu for a certain number of steps, even if the values in these cells change in the meantime.

When a move $\mu(\alpha, i, j_1, j_2)$ is performed at step h, the tabu table entry $t(\alpha, i, j_2, j_1)$ is set to $h + d$, where d is the current tabu tenure. For a right-shift move $\bar{\mu}(\alpha, i, w)$, the entry $t(\alpha, i, (n - w) \bmod n)$ is set accordingly.

The tabu tenure ranges dynamically between *maxTabu* and *minTabu* steps. On every non-improving step, the tenure is increased by one, up to *maxTabu*. On every improving step, the value is decreased by one, down to *minTabu*. In our experiments, *maxTabu* was set to 10, and *minTabu* to 2. These values were taken from [14].

In order to escape the area of a local minimum without a complete restart, we added a perturbation component that alters parts of the current assignment. When the algorithm fails to improve on the current best assignment for *maxStable* steps, the current assignment is perturbed by right-shifting a random row from each rectangle (except the control rectangle) by a random amount. In our experiments, *maxStable* was set to 150, since initial tests showed that the search rarely progressed further for higher values.

The perturbation makes a major step away from the current assignment, often allowing the algorithm to reach a different part of the search space, so that it can escape the area of local minimum where it got stuck. For each iteration of the search, the algorithm can make *maxPert* such perturbations, with *maxPert* set to 2 in our experiments.

If after *maxPert* perturbations the *stableIter* counter again reaches the value *maxStable*, the algorithm restarts the search with a new initial assignment. In total *maxIter* iterations of the search run are performed, with *maxIter* set to 10 in our experiments. (A relatively small value of *maxIter* was chosen in order to keep the amount of CPU time required for our experiments manageable.)

4.4 Seeding the Search

Initially, we used random permutations of $\{0 \ldots n-1\}$ to initialise the rows of the rectangles. However, this gave unsatisfactory results, with the search often starting with a very high violation count. The search usually became repeatedly stuck in local minima, failing to reach a solution despite the perturbation moves that allowed the search to escape the area of a local minimum.

Therefore, we also used constructive heuristics to improve the seeding, similar to Dotú and Van Hentenryck [4]. Our first seeding heuristic uses the MOLR construction described Section 3.1. We select parameters for the heuristic with the same value of n and where possible at least as many rectangles and rows as the final MOLR instance we desire, filling in any missing rows with random permutations of $\{0 \ldots n-1\}$.

Our second seeding heuristic uses the column problem construction described in Section 3.2. In this case we take a near-solution to the corresponding column problem

(also found with local search) and use it to construct a near-solution to the MOLR problem, which is then used as the seed.

The performance of the different seeding heuristics is discussed in Section 5.

4.5 Local Search for the Column Problem

We model the column problem corresponding to r MOLR of order $m \times n$ using an array C of dimension $m \times r$, with elements taking values from $\{0 \ldots n-1\}$. Note that $C(i,\alpha)$ actually corresponds to $L_\alpha(i,0)$ from the MOLR problem. We also associate with each column α the set of numbers $\mathcal{U}_\alpha \subset \{0 \ldots n-1\}$ that are unused in that column.

We initialise each column such that each cell in that column contains a different value, thus fulfilling the alldifferent constraint (7). If we allow only exchanges of values between cells within a column, or between a cell and the set of unused values for that column, constraint (7) will always be observed. Therefore, the evaluation function uses only the number of violations of constraint (5):

$$f_c(\sigma) = \sum_{i_1=0}^{m-2} \sum_{i_2=i_1+1}^{m-1} \sum_{x=0}^{n-1} max(0, \#(\sigma, i_1, i_2, x))$$

$$\text{with} \quad \#(\sigma, i_1, i_2, x) = |\{\alpha \in \{1 \ldots r\} | (-C(i_2,\alpha) + C(i_1,\alpha))_\sigma = x)\}| - 1$$

where the arithmetic is done in the chosen group of order n.

The search uses two types of moves: swapping the values of two cells in a column, and swapping the value of a cell with a value from the set of unused values in its column:

$$\mu_c(\alpha, i_1, i_2) = [C(i_1,\alpha) \leftrightarrow C(i_2,\alpha)]$$
$$\text{and} \quad \mu'_c(\alpha, i, e) = [C(i,\alpha) \leftrightarrow e \in \mathcal{U}_\alpha]$$

Again, the neighbourhood is restricted to moves involving cells in violation. Formally, it is defined as:

$$\mathcal{S} = \mathcal{S}_s \cup \mathcal{S}_e$$

where

$$\mathcal{S}_s = \{\mu_c(\alpha, i_1, i_2) | \exists i \in \{0 \ldots m-1\} \; : \; (i \neq i_1 \wedge \#(\sigma, i_1, i, -C(i,\alpha) + C(i_1,\alpha)) > 1) \\ \vee \; (i \neq i_2 \wedge \#(\sigma, i_2, i, -C(i,\alpha) + C(i_2,\alpha)) > 1)\}$$

$$\text{and} \quad \mathcal{S}_e = \{\mu'_c(\alpha, i_1, e) | \exists i_2 \in \{0 \ldots m-1\} : i_2 \neq i_1 \wedge \\ \#(\sigma, i_1, i_2, -C(i_2,\alpha) + C(i_1,\alpha)) > 1 \wedge e \in \mathcal{U}_\alpha\}$$

Should there be no move that will improve the evaluation, the algorithm will, as for the MOLR problem, choose with probability p_1 a move that least increases the violation count; with probability p_2, it will swap two randomly chosen values within a randomly chosen column; and with probability p_3, it will down-shift a randomly chosen column by a random amount.

The tabu search algorithm is the same as for the MOLR problem. In our experiments, we also set the parameters to the same values, with two exceptions. The first difference is that *maxPert* is set to zero; i.e. every time the stable iteration counter reaches

the value *maxStable*, the search restarts with a random initialisation. Secondly, *maxIter* is set to 20, since the column problem model is much smaller than the corresponding MOLR model and the search progresses faster.

5 Results

We explored several approaches to solving the MOLR problem, eventually trying instances up to $n = 20$:

1. **Constructive backtracking search with symmetry breaking on the full MOLR problem.** This was implemented in ECL^{i}PS^e [21] using the SBDD symmetry library described in [8]. While this was able to completely solve (and enumerate) instances with $n = 6$, non-trivial instances with $n \geq 10$ seem beyond the reach of this kind of approach at this time.
2. **Local search on the full MOLR problem**, as discussed in Section 4. This was also implemented in ECL^{i}PS^e, and was able to solve some previously open instances, but it was not particularly effective and was later surpassed by other approaches.
3. **Construction**, as discussed in Section 3.1. This was implemented in GAP [7], with an ECL^{i}PS^e wrapper to allow integration with the other approaches. This approach generally provided a reasonably good solution of the form $m-1$ MOLR of order $m \times n$ for each value of n. These solutions were later surpassed by other techniques, but of course it is a fast way to obtain some MOLR for problem instances that are too large to tackle with search.
4. **Local search on the full MOLR problem, seeded with a constructed near-solution**, as discussed in Section 4.4. This was much more effective than starting with a random seed; typically the seed for an instance had fewer violations than the best assignment found without the seed, and enabled a number of new instances to be solved. A number of these (notably for $n = 12$ and $n = 15$) have not been matched by any other technique.
   In section Section 3.1, we presented a construction that allowed us to generate $p^e - 1$ MOLR of order $p^e \times n$. In our experiments, we found that a good seeding heuristic for finding r MOLR of order $m \times n$ is to set p to the prime number nearest to $max(r,m)$ and leave n un-factorised, so that $e = 1$. Only for some instances where a factorisation of n exists such that r, m and p^e are close together did such a seed yield a better result (indicated by $4'$ in Table 1) — usually factorising n was worse.
5. **Constructive backtracking search on the column problem.** The most effective variant we tried was to simply assign variables random values from their domain, giving up if no solution was found after 60 seconds of CPU time. We tried two static variable ordering heuristics: rectangle-by-rectangle (good for instances with few rectangles) and row-by-row (good for instances with few rows). We also tried using groups other than the integers modulo n for the arithmetic, which sometimes yielded superior results (indicated by a $5'$ in Table 1). Solutions, when found, were generally found in just a few seconds. This was one of the most effective of the methods we tried for finding MOLR solutions; there were very few instances solved by other methods that were not solved by this one.

6. **Local search on the column problem**, as discussed in Section 4.5. This was the other very effective method, able to find a number of solutions not found by the constructive backtracking approach, but also failing to find some solutions that were found by that approach. As with the constructive approach above, we tried using different groups for the arithmetic, and sometimes this yielded better results (indicated by a $6'$ in Table 1). Solutions, when found, were generally found in just a few seconds; with our parameter settings even the largest instances ran for no more than a few minutes if no solution was found. One of the reasons for trying this approach is that even when it could not find a solution, any near-solutions it found could be used as seeds for local search on the full problem.
7. **Local search on the full MOLR problem, seeded with a near-solution from the column problem.** This approach turned out to be something of a disappointment, failing to find solutions (only near-solutions) for any instances we tried that were not solved by local search on the column problem. It is possible that a near-solution to the column problem is a poor heuristic choice for seeding the full problem.

A summary of the best results for $n \leq 20$ is given in Table 1. *LB* is the best known lower bound; *Method* indicates which of the above methods was able to solve that instance. We have included previously known results, from the extensively-studied MOLS problem [2–II.2] and from [22]. We have omitted those values of n for which a complete set of MOLS is known, since all such instances are solved by that set of MOLS. We have also omitted results for $m = 2$ since there is a trivial construction for a complete set of MOLR in this case. Finally we have omitted listing the methods for those entries that are dominated by another entry (i.e. the latter has more rectangles and/or more rows), except where it is useful to show how close our techniques are to matching the MOLS results of [2–II.2].

An expanded set of results including solutions can be found on the web [10].

We suspect that our failure to match most of the MOLS results is at least in part due to the fact that our most successful techniques can only find solutions of a certain form, and for some instances there are no solutions of this form. For example, a complete search of the column problem found no solution for 2 MOLS of order 10, even though a solution to the full problem exists. While we are very pleased with the results that we have obtained, for the most part we do not know how far from optimal they are. Complete search is currently out of the question for all but the smallest instances, and even for the well-studied MOLS problem very few good upper bounds are known. We do, however, expect that our results can be significantly improved upon, particularly for larger instances.

As shown in Section 2.2, MOLR solutions can be used to construct solutions to the social golfer problem. Using the new results in Table 1 we were able to construct solutions to the 43 instances listed in Table 2. The *gain* indicates the number of extra rounds achieved over the previously best known result from any source (RBIBD, MOLS, constructions, constraint programming, etc.). Solutions to much larger instances of the social golfer problem remain within easy reach using the techniques we have presented here; we merely had to stop somewhere. A full table of results for the social golfer problem from all sources can be found on the web [11].

Table 1. Summary of MOLR results: lower bounds for $N(m,n)$

	n													
	6		10		12		14		15		18		20	
	LB	Method	LB	Method	LB	Method	LB	Method	LB	Method	LB	Method	LB	Method
3	4		8		11		12		14		16		19	
4	4		8		11		12	5′,6	14	5	16	4,5	19	5′
5	4	1,2,4,5,6	8	5	11		11	5′,6′	11	5,6	14	5,6	16	5,6′
6	1	(trivial)	6		11	5′	10	5′	10		13	5′	15	6′
7			6	5,6	8		8	4,5,6	10		12	5′	13	6
8			4		8	4′	7	5,6	10		10	4,5,6	11	5,6
9			4	[22],5	7	4′	6	4,5,6	10		9	6′	10	4,5
10			2	[2]	5		5		10		8	6′	9	5,6
m 11					5		5	4	10	4	7	5′	8	4,5′,6
12					5	[2],5′	4	5	4	5,6	6	5	7	5
13							3	5	4		5	5,6	6	
14							3	[2]	4		4		6	5′
15									4	[2]	4		5	5,6
16											4	5′	4	
17											3	5,6	4	5,6′
18											3	[2]	4	
19													4	
20													4	[2]

Table 2. New solutions for the social golfer problem (g-s-w)

Instance	Gain	Instance	Gain	Instance	Gain	Instance	Gain	Instance	Gain
10-6-7	+1	14-5-12	+5	15-6-11	+6	18-5-16	+7	20-6-16	+10
10-7-7	+2	14-6-11	+4	15-7-11	+6	18-6-15	+6	20-7-14	+9
10-8-5	+1	14-7-10	+2	15-9-11	+6	18-7-13	+4	20-8-12	+7
10-9-5	+1	14-8-8	+4	15-10-11	+6	18-8-11	+2	20-9-11	+6
		14-9-7	+3	15-11-11	+6	18-9-11	+1	20-10-11	+5
12-7-9	+3	14-10-6	+2			18-10-9	+5	20-11-9	+4
12-8-9	+3	14-11-6	+2			18-11-8	+4	20-12-8	+3
12-9-8	+2	14-12-5	+1			18-12-7	+3	20-13-7	+2
						18-13-7	+3	20-14-7	+2
						18-14-6	+2	20-15-6	+1
						18-15-5	+1	20-16-6	+1
						18-16-5	+1		

6 Conclusions and Future Work

We have shown that by solving a reduced problem, one can construct good solutions to the MOLR problem, which can be used to construct solutions to various other problems of interest. In particular, we have shown how generalisations of MOLS-based constructions can use these solutions to yield improved solutions to 43 instances of the social golfer problem.

We have also confirmed Dotú and Van Hentenryck's result [4] that seeding a local search algorithm with heuristically-constructed solutions with low violation counts can dramatically improve results on this kind of combinatorial problem, where the local search landscape contains many local minima and it is in general hard to progress towards a global optimum. However, we have found that this is not always a benefit: the wrong kind of construction can yield a seed which has a good initial violation count, but starts the search in a local minimum from which it is almost impossible to escape.

The MOLR and social golfer problems are still both far from solved. It would be interesting to see how far our MOLR results can be improved, and what other techniques can be used to construct new solutions to the social golfer problem. One thing we plan to investigate further is seeding a local search for the social golfer problem with a constructed assignment based on a near-solution of the MOLR problem. Early experiments with this approach have yielded a solution to the previously unsolved 14-8-9 instance. We also plan to continue to investigate construction techniques for the MOLR and social golfer problems.

Acknowledgments

We would like to thank Jonathan Lever for his local search code for the social golfers problem that we adapted for the MOLR and column problems, and Ian Wanless for pointing out useful references and for providing encouragement to work on the MOLR problem. We would also like to thank Pascal Van Hentenryck and `mathtalk-ga` for interesting discussions on constructions for the social golfer problem. Finally, we would like to thank the anonymous reviewers for their feedback and suggestions.

References

1. Nicolas Barnier and Pascal Brisset. Solving the Kirkman's Schoolgirl Problem in a few seconds. In P. Van Hentenryck, editor, *CP 2002: Proc. of the 8th Int. Conf. on Principles and Practice of Constraint Programming*, LNCS 2470, pages 477–491. Springer-Verlag, 2002.
2. C.H. Colbourn and J.H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs*. CRC Press, Rockville, Maryland, USA, 1996.
3. Ivan B. Djordjevic and Bane Vasic. LDPC codes for long haul optical communications based on high-girth designs. *Journal of Optical Communications*, 24(3):94–96, 2003.
4. Iván Dotú and Pascal Van Hentenryck. Scheduling social golfers locally. In Roman Barták and Michela Milano, editors, *CP-AI-OR 2005: Proceedings of the 2nd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, LNCS 3524, pages 155–167, 2005.
5. Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry breaking. In Toby Walsh, editor, *CP 2001: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, LNCS 2239, pages 93–107, 2001.
6. M.F. Franklin. Triples of almost orthogonal 10×10 latin squares useful in experimental design. *Ars Combinatoria*, 17:141–146, 1984.
7. The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.3*, 2002. (`http://www.gap-system.org`).

8. Ian P. Gent, Warwick Harvey, Tom Kelsey, and Steve Linton. Generic SBDD using computational group theory. In Francesca Rossi, editor, *CP 2003: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, LNCS 2833, pages 333–347. Springer-Verlag, 2003.
9. Ian P. Gent, Toby Walsh, and Bart Selman. CSPLib: a problem library for constraints. http://csplib.org/.
10. Warwick Harvey. Warwick's results page for the MOLR problem. `http://www.icparc.ic.ac.uk/~wh/molr`.
11. Warwick Harvey. Warwick's results page for the social golfer problem. `http://www.icparc.ic.ac.uk/~wh/golf`.
12. Warwick Harvey. Symmetry breaking and the social golfer problem. In Pierre Flener and Justin Pearson, editors, *Proc. SymCon-01: Symmetry in Constraints*, pages 9–16, 2001.
13. `mathtalk-ga`. Answer to "Unique combinations of 4 numbers between 1 to N". *Google Answers*, 2005. `http://answers.google.com/answers/threadview?id=274891`.
14. Laurent Michel and Pascal Van Hentenryck. A simple tabu search for warehouse location. *European Journal of Operations Research*, 157(3):576–591, 2004.
15. Gary L. Mullen and Jau-Shyong Shiue. A simple construction for orthogonal latin rectangles. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 9:161–166, 1991.
16. Steven Prestwich. Randomised backtracking for linear pseudo-boolean constraint problems. In Narendra Jussien and François Laburthe, editors, *Proc. of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 7–19, Le Croisic, France, March, 25–27 2002.
17. Jean-François Puget. Symmetry breaking revisited. In Pascal Van Hentenryck, editor, *CP 2002: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, LNCS 2470, pages 446–461. Springer-Verlag, 2002.
18. V. K. Sharma and M. N. Das. On resolvable incomplete block designs. *Austral. J. Statist.*, 27(3):298–302, 1985.
19. Barbara M. Smith. Reducing symmetry in a combinatorial design problem. In *CPAIOR'01: Proc. of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 351–359, April 2001.
20. D. R. Stinson, R. Wei, and L. Zhu. New constructions for perfect hash families and related structures using combinatorial designs and codes. *Journal of Combinatorial Designs*, 8(3):189–200, 2000.
21. Mark G. Wallace, Stefano Novello, and Joachim Schimpf. ECLiPSe : A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, May 1997.
22. Ian M. Wanless. Answers to questions by Dénes on latin power sets. *Europ. J. Combinatorics*, 22:1009–1020, 2001.