The Construction of Subsquare Free Latin Squares by Simulated Annealing

J.R. Elliott and P.B. Gibbons
Department of Computer Science
University of Auckland
New Zealand

Abstract

A simulated annealing algorithm is used in the construction of subsquare free Latin squares. The algorithm is described and experience from its application is reported. Results obtained include the construction of subsquare free Latin squares of orders *16* and *18*, the smallest orders for which existence of such squares was previously in doubt.

Key words

Latin square, subsquare, simulated annealing

1. Introduction

A *Latin square* of side n is an $n \times n$ matrix with entries from the set of n integers $\{1, 2, \ldots, n\}$ and the property that each integer from the set $\{1, 2, \ldots, n\}$ occurs exactly once in each row and column. A *subsquare* of side k $(1 \le k \le n)$ of a Latin square L is a triple (R, C, E), $|R| = |C| = |E| = k$, where $R = \{r_1, r_2, \ldots, r_k\}$ is a set of row names, $C = \{c_1, c_2, \ldots, c_k\}$ is a set of column names, and $E = \{e_1, e_2, \ldots, e_k\}$ is a set of entries from L. The triple (R, C, E) satisfies the property that the entry in row r_i and column c_j $(1 \le i, j \le k)$ is always contained in E. In other words, a *subsquare* of side k is a Latin square of order k resulting from the deletion of $n-k$ rows and columns form a Latin square of order n. A *proper* subsquare of side k has $1 < k < n$.

Latin squares have been widely studied (see, for example, Denes & Keedwell [4]) and have close connections with groups and quasigroups. In this paper we focus our attention on the

construction of Latin squares with no proper subsquares, where a number of open existence questions remain.

The first major step in determining the spectrum of Latin squares with no proper subsquares was made by Heinrich [7] who managed to construct subsquare free Latin squares (SFLS's) of order $n = pq$, where p and q are distinct primes, and $n \neq 6$. Her method was extended and generalized by Andersen and Mendelsohn [2], who showed that such squares exist for all n not of the form $n = 2^a3^b$. When n *is* of the form 2^a3^b it is known that SFLS's do *not* exist for $n = 4, 6$, and that they *do* exist for $n = 8$ (see [5], [7], and [10]) and $n = 12$ (see [6]).

Mendelsohn and Rosa [14] have investigated a relationship between one-factorizations and SFLS's. Their work adds to the spectrum of orders of the form $n = 2^a3^b$ for which SFLS's are known to exist by showing that there exist SFLS's of orders $3, 9, 27, 81$, and 243, but for no other known orders $n = 3^b \leq 3^{12}$.

From the above we see that the first unsolved cases for SFLS's are currently $n = 16, 18$ and 24.

Highly relevant to our study of SFLS's is work that has been carried out on the existence of Latin squares with no subsquares of side 2. Work by Kotzig, Lindner and Rosa [9], McLeish [13], and Kotzig and Turgeon [10] has established that such 2×2 subsquare free Latin squares (N2LS's) exist if and only if $n \neq 2, 4$. Although not sufficient to guarantee that a Latin square is completely subsquare free, we have found that the 2×2 subsquare free property (N_2) provides a very strong screening in the search for SFLS's.

In related work, Gibbons and Mendelsohn [6] used a computer search to construct a SFLS of order 12. The constructive backtracking approach used in that paper was not feasible in the search for SFLS's of order 16, 18 and 24. Instead the authors of this paper trialled a probabilistic, incremental change method, *simulated annealing*, and used it successfully to construct SFLS's of orders 16 and 18. In this paper we describe that method and report on our experience from its use. We believe the results provide an endorsement of the effectiveness of this strategy in coping with certain types of existence problems in combinatorics.

2. The simulated annealing method

The simulated annealing method originated in the early 1980's when Kirkpatrick, Gelatt and Vecchi [8] and Cerny [3] independently explored an analogy between the physical annealing process of solids and the task of solving large combinatorial optimization problems. A complete description of this process is provided by Aarts & Korst [1] or Laarhoven [11]. We

provide here a brief summary from these two excellent references.

Annealing, in condensed matter physics, refers to a thermal process for obtaining low energy states of a solid in a heat bath. In this process, the solid is first heated to melting temperature and then slowly cooled until the low-energy ground state is reached. If the initial temperature is not sufficiently high or the cooling is carried out too quickly, the solid will freeze into a meta-stable state rather than the ground state. The converse of annealing is *quenching* where the temperature of the heat bath is instantaneously lowered, resulting in a meta-stable state.

The physical annealing process can be modelled successfully by using computer simulation methods. In one of these approaches, Metropolis, Rosenbluth & Rosenbluth, Teller & Teller [15] in 1953 introduced a simple Monte Carlo algorithm which generates a sequence of states in the solid in the following way. Given a current state i of the solid, characterized by the positions of its particles, with energy E_i, a subsequent state j, with energy E_j, is generated by applying a small distortion to the current state, for example by the displacement of a particle. If the energy difference, $E_j - E_i$ is ≤ 0, then the state j is accepted as the new current state. If the difference is > 0, then the state j is accepted with probability

$$exp\ ((E_i - E_j)/k_B T)$$

where T denotes the temperature of the heat bath and k_B is a physical constant known as the Boltzmann constant. This acceptance rule is known as the *Metropolis* criterion and the associated algorithm is known as the *Metropolis algorithm*.

If the lowering of the temperature is done sufficiently slowly, the solid can reach thermal equilibrium at each temperature. In the Metropolis algorithm this as achieved by generating a large number of transitions at a given temperature value. In thermal equilibrium the probability of occurrence of a solid being in a state i with energy E_i at temperature T is given by the Boltzmann distribution

$$P_T[X = i] = (exp(- E_i / k_B T)) \ / \ Z(T)$$

where X is a stochastic variable denoting the current state of the solid. $Z(T)$ is the partition function which is defined as

$$Z(T) = \Sigma\ exp(- E_j / k_B T)$$

where the summation extends over all possible states.

With a combinatorial optimization problem we can simulate this annealing process by making the following correspondences:

(1) Feasible solutions correspond to states of the solid.
(2) The cost *f(i)* of a feasible solution *i* corresponds to the energy of the corresponding state.
(3) A randomly chosen feasible solution corresponds to the state of the solid after it has been heated to melting point.
(4) An elementary change to a feasible solution corresponds to a slight distortion of the state of the solid.
(5) An introduced control parameter *T* corresponds to the temperature of the heat bath.

The simulated annealing algorithm can now be viewed as an iteration of Metropolis algorithms, evaluated at decreasing values of the control parameter. Initially the control parameter is given a large value, and, starting with an initial randomly chosen feasible solution, a sequence (or Markov chain) of trials is generated. In each trial, a configuration *j* is generated by applying a random elementary change to the current configuration *i* (the set of configurations attainable by such changes is called the neighbourhood of *j*). If $\Delta_{ij} = f(j) - f(i)$, then the probability of configuration *j* being the next configuration in the sequence is *1* if $\Delta_{ij} \leq 0$ (for a minimization problem), and $exp(-\Delta_{ij} / T)$ if $\Delta_{ij} > 0$. The control parameter is lowered in steps with the system being allowed to approach equilibrium through the sequence of trials at each step. After an appropriate stopping condition is met, the final configuration is taken as the solution of the problem at hand. There are a variety of possible stopping conditions. A common one is to terminate the algorithm when the observed improvement in cost over a number of consecutive Markov chains is small.

Suppose T_k is the value of the control parameter and L_k the length of the Markov chain generated at the *k*th iteration of the Metropolis algorithm. Suppose *random[0,1)* returns a random number generated from a uniform distribution on the interval *[0,1)*. Then a pseudo-Pascal description of the simulated annealing algorithm is as follows:

```
procedure Simulated_Annealing;
begin
    Set initial values for i (state), T (temperature) and L (Markov chain length);
    k := 0;
    repeat
        repeat L times
        begin
            Generate j from neighbourhood of i;
            if Δij ≤ 0 or random[0,1) < exp(- Δij/ T)
            then i := j;
        end;
        k := k+1;
        Update L for kth iteration;
        Update (reduce) T;
    until stop condition is met
end
```

The performance of this algorithm will be influenced by the following factors:

(1) The initial value of T.
(2) The method for reducing the value of T after each sequence of trials.
(3) The length of Markov chain for each value of T.
(4) The choice of stopping condition.

The specification of these parameters constitutes a *cooling schedule.*

3. Application to Latin square generation

Since its initial investigation in the early 1980's, simulated annealing has been applied to a wide variety of combinatorial optimization problems, particularly *NP*-hard problems. Such problems include travelling salesman, graph partitioning, matching, colouring, scheduling, VLSI design, image processing, and facilities layout. A comprehensive survey of applications is detailed in [1].

In these optimization problems the effectiveness of the simulated annealing approach is measured by the closeness of the final solution to the optimal solution (the error), and the running time of the algorithm. With combinatorial existence problems, such as the construction of subsquare free Latin squares, our aim is to construct optimal solutions rather

than approximations to them. With this modified goal, our measure of effectiveness will be based on the running time of the algorithm and the strike rate for obtaining optimal solutions. However, it would be unreasonable to expect such running times to be polynomially bounded.

Although the hill-climbing method has been applied to existence and enumeration problems in combinatorics (see, for example, Stinson [16]), it appears that simulated annealing has not been widely used. Mathon [12] has used the method successfully to construct balanced incomplete block designs, and in this paper we adapt his method to the construction of subsquare free Latin squares.

Gibbons and Mendelsohn [6] observed that applying the N_2 condition provides a useful screening check in the search for subsquare free Latin squares - the check is easily applied and there is a high chance that an N2LS will in fact be a SFLS. The goal of a simulated annealing algorithm will therefore be to start with an initial random Latin square, and then perform elementary changes in an attempt to reduce the number of 2×2 subsquares. That is, our state will be a Latin square, and our cost function will be the number of 2×2 subsquares contained in it. We apply an elementary change which maintains the Latin square property, but which hopefully destroys 2×2 subsquares. This elementary change is described as follows.

Suppose x and y are elements, and r a row, of a Latin square L of side n. Define the (r, a, b)-cycle of L as the sequence $((r_1, c_1), (r_2, c_2), \ldots, (r_m, c_m))$, where $r_1 = r, r_2, \ldots, r_m$ are rows of L, $c_1, c_2, \ldots, c_m$ are columns of L, and

(1) If i is even, $L(r_i, c_i) = y$ and $c_i = c_{i-1}$

(2) If i is odd, $L(r_i, c_i) = x$ and $r_i = r_{i-1}$ (for $i > 1$), $r_1 = r_m$

For example, suppose L is the following Latin square of side 8:

8	2	3	4	5	6	7	**1**
2	3	1	5	6	7	8	4
3	5	4	**8**	7	**1**	6	2
4	6	8	3	1	5	2	7
5	**8**	7	**1**	3	2	4	6
6	7	5	2	8	4	1	3
7	**1**	2	6	4	3	5	**8**
1	4	6	7	2	**8**	3	5

In this square the elements of the $(1, 1, 8)$-cycle, $((1,8), (7,8), (7,2), (5, 2), (5, 4), (3, 4), (3, 6),$

(8, 6), (8, 1), (1, 1)), have been highlighted. There is only one other cycle involving the elements *1* and *8*, and that is the *(2, 1, 8)*-cycle *((2, 3), (4, 3), (4, 5), (6, 5), (6, 7), (2, 7))*.

From the properties of a Latin square, it is clear that such cycles are well defined. More importantly, if the elements making up a cycle in a Latin square are rotated, then a new (in general non-isomorphic) Latin square will be obtained. Such a rotation can therefore be used as a transition operation in a simulated annealing algorithm. Since our goal is to alter the number of subsquares (hopefully downwards), we should choose cycles which pass through an element of a 2×2 subsquare. For example, the above square contains exactly 1 2×2 subsquare, viz. the subsquare with *(R, C, E) = ((4, 5), (4, 5), (1, 3))*. The *(1, 1, 8)*-cycle passes through the element *1* at position *(5, 4)* in this square (and no other elements). Therefore, if we rotate this cycle, the subsquare will be destroyed. In general other subsquares may be created by this transformation. Also, if the cycle passes through more than one element of the subsquare, a new subsquare in the same position will be created.

Applying a rotation to the *(1, 1, 8)*-cycle we produce the following Latin square which is N_2 (and also subsquare free):

1	2	3	4	5	6	7	8
2	3	1	5	6	7	8	4
3	5	4	1	7	8	6	2
4	6	8	3	1	5	2	7
5	1	7	8	3	2	4	6
6	7	5	2	8	4	1	3
7	8	2	6	4	3	5	1
8	4	6	7	2	1	3	5

We now have the basis for a simulated annealing algorithm in which the state corresponds to a Latin square and the energy level corresponds to the number of subsquares of side 2. For our transition operation we select a random subsquare $S = ((r_1,r_2), (c_1,c_2), (e_1,e_2))$ of side 2. We then select a random element $x \neq e_1,e_2$, a random element $y \in \{e_1,e_2\}$ and perform a rotation on the (r_1, x, y)-cycle.

For our temperature decrement function we chose to multiply the temperature by a constant $\alpha < 1$, i.e. $T_{k+1} = \alpha T_k$. We shall call α the *control decrement*. Also, we kept the length of the Markov chain the same for each new value of the control temperature. For our stopping condition we chose to terminate the algorithm once the cost function of the best solution

obtained in the last Markov chain remains unchanged for a number of consecutive chains. The number of such chains we shall call the *freezing factor*.

4. Results

The above described simulated annealing algorithm was first implemented in the language *C* on Apple Macintosh computers, and then later on faster DecStation 2100's running under the UNIX operating system. Initially we tested the algorithm with Latin squares of orders *8* and *12* in an attempt to get a feel for the best type of cooling schedule to employ. We also wished to see how effective the N_2 condition is in producing completely subsquare free Latin squares.

Although Aarts & Korst [1] and others have shown that it is possible in theory to set a cooling schedule which guarantees production of an optimal solution (in this case an N2LS), such a schedule in practice usually requires a number of transitions that is typically larger than the size of the solution space, leading to an expected exponential-time execution of the algorithm. Instead we therefore ran a number of iterations of a cooling schedule which requires a polynomial number of transitions, and recorded the number of N2LS's generated. Building on Mathon's experience with annealing algorithms for building block designs, we used the following cooling schedule:

Initial temperature:	*1.0*
Control decrement:	*0.9995*
Markov chain length:	*350*
Freezing factor:	*12*

Order *8* N2LS's were constructed very easily. In the order *12* case, on a DecStation 2100 we were able to generate *33* N2LS's over a *12* hour period[1]. Of these *33* squares, one was found to contain a subsquare of order 3. The remaining *32* were found to be completely subsquare free.

The success of this method with the order *12* case encouraged us to go after the order *16* and order *18* cases. Although N2LS's of these orders are known to exist, the existence of SFLS's of these orders had not yet been established. Running the algorithm on a DecStation 2100 with the above cooling schedule, we were able to generate 2 N2LS's of order *16* over a *48* hour period. Both of these squares were found to be completely subsquare free. In the order *18* case we were able to generate *1* N2LS, which turned out to be subsquare free, over a 7-day period. This order *18* SFLS is displayed and analysed in the Appendix, along with one of the order

[1] All times mentioned are processing times rather than elapsed times

16 SFLS's.

The next unknown case is of order *24* (= $2^3 3^1$). Currently we have been unable to obtain an N2LS of this order, the best square produced containing *3* subsquares of order 2. In order to fine tune the algorithm to have the best chance of finding an N2LS we decided to investigate the effect of varying the cooling schedule in the search for N2LS's of order *12*. The following results were obtained.

We began by studying the effect of changing the initial value of the control parameter (temperature), while holding the control decrement, Markov chain length, and freezing factor constant at *0.9995*, *350*, and *12* respectively. The results in this and most of the following experiments were obtained from runs over a *12* hour period:

Initial Temp	Initial Acceptance Ratio	# trials	N2LS's Generated	Strike Rate	Average Drop Out Temp
1.0	13%	124	29	23%	0.6957
2.0	48%	45	13	29%	0.7287
3.0	56%	33	14	42%	0.8034
4.0	78%	27	14	52%	0.7633
5.0	82%	24	13	54%	0.7796
6.0	85%	21	9	43%	0.7117
7.0	89%	20	7	35%	0.7289

From this table it can be seen that, with the other parameters in the cooling schedule held constant at the given values, it does not appear to pay to start at a high temperature both in terms of the number of N2LS's generated in a set period, and the strike rate ((number of N2LS's generated) / (number of trials)). This is surprising given the recommendation, for example in [1], to begin with a temperature which results in an acceptance ratio of close to *1*. It may be, of course, that altering some of the other parameter values would improve the algorithm's performance with the higher starting temperatures. This needs to be the subject of a further investigation. In the meantime we decided to use a starting temperature of *1.0* for the remaining experiments.

We next studied the effect of varying the value of the control decrement, while holding the chain length and freezing factor constant at *350* and *12* respectively.

Control Decrement	# trials	N2LS's	Strike Rate	Average Drop Out Temp
0.95	2939	32	1%	0.2518
0.995	707	47	7%	0.5266
0.9995	124	33	27%	0.6929
0.99995	43	40	93%	0.9039

In terms of strike rate, the above results illustrate the tradeoff between the value of the decrement of the control parameter and the length of the Markov chain. Large decrements in the control parameter will require longer Markov chain lengths in order to restore quasi equilibrium at the next value of the control parameter. It is clear that with the chain length held constant we have a better chance of reaching equilibrium with a smaller decrement value, and hence a better chance of obtaining an optimal solution. However, in terms of the number of N2LS's generated, the results are less conclusive. The *47* N2LS's generated with a control decrement of *0.995* goes against the trend of more squares generated for higher values of the control decrement. The low strike rate achieved for this value lead us to reject it in favour of *0.99995* for the remaining experiments.

The next parameter to be varied was the Markov chain length, with the control decrement and freezing factor held constant at *0.99995* and *12* respectively.

Chain Length	# trials	N2LS's	Strike Rate	Average Drop Out Temp
100	138	35	25%	0.8921
200	56	27	48%	0.8665
300	39	32	82%	0.8726
400	28	24	86%	0.8687
500	39	37	95%	0.9239

These results again illustrate the fact that longer Markov chain lengths will give the algorithm a better chance of reaching equilibrium at each new value of the control parameter. However, as with the previous table, the results are less conclusive in suggesting the best chain length to use to maximise the numbers of squares obtained in a set time period. In [1] it suggested that the length should be chosen so as to give the algorithm a sufficiently large probability of visiting at least a major part of the neighbourhood of a given feasible solution. This is quantified by taking the length equal to the size of the neighbourhood of a given

feasible solution. In our case the neighbourhood depends on the number of 2 ×2 subsquares in the current square, and this number decreases as the algorithm progresses. The initial random Latin square contains 32 2 ×2 subsquares on average, resulting in a neighbourhood size of *(12-2)*2*32 = 640*. However, the number of subsquares rapidly decreases to less than *10* after the start of the algorithm, resulting in a typical neighbourhood size of *200*. In the interests of achieving a reasonably high strike rate we opted to continue with a chain length of *350* for the remaining experiment. However it is clear that more tests are required here, including the option of varying the chain length at each new value of the control parameter depending on the size of the neighbourhood of the first feasible solution in the chain.

Our final experiment tested the effect of varying the freezing factor with the control decrement and chain length held constant at *0.99995* and *500* respectively. The results, obtained over a *24* hour period, were as follows:

Freezing Factor	# trials	N2LS's	Strike Rate Generated	Average Drop Out Temp
4	468	41	9%	0.9865
6	105	56	53%	0.9414
8	59	46	78%	0.8985
10	63	56	89%	0.9056
12	55	53	96%	0.8921

It is clear that as the freezing factor increases (i.e. we continue the annealing process longer) the strike rate should increase and the number of trials per unit time should decrease. The average drop-out temperature should also decrease with increasing freezing factor values. The above results conform to this expectation. In terms of the number of optimal solutions per unit time the best results were for freezing factors of *6* and *10*. Certainly there would be no point in continuing beyond *12* since the strike rate is very close to *100%* and the number of trials will decrease. Because of the high strike rate we favoured a freezing factor of *10* or *12*.

What is interesting about this table, and others in this section, is the fact that we are getting good results at high drop-out temperatures. The results in this latest experiment were obtained within a relatively narrow temperature interval of around *0.11*. However, we must remember that we have begun at a temperature which results in an initial acceptance ratio of around *13%*, already well below the recommended value of around *90%*. The resulting probability of accepting uphill moves must be enough both to provide a good chance of getting out of local minima and of directing the search toward global optima.

5. Conclusions

In this paper we have applied the well-known simulated annealing method to an optimisation problem where we require optimal solutions rather than close approximations to them. We are assisted by the fact that we can easily detect an optimal solution when we have found one.

The problem in question is that of generating subsquare free Latin squares, and the method was successful in generating such squares of orders *16* and *18*, the smallest orders for which existence of such squares was previously in doubt. The algorithm has so far been unsuccessful in generating a SFLS of order 24, the next order for which existence has not yet been decided.

Some experimentation was carried out with the generation of SFLS's of order *12* in order to try to determine the combination of parameter values most likely to generate a solution in a set period of time. While giving some pointers to likely parameter values, it is clear that a more elaborate statistical experiment is required for us to more accurately predict the combination of cooling schedule parameters most likely to product optimal solutions in a set period of time. Such an analysis would be valuable since the method holds some promise of solving other combinatorial existence problems of this type.

6. Acknowledgements

The authors are grateful to R.A. Mathon for allowing us to experiment with his simulated annealing algorithm for the construction of block designs. Many of his ideas were incorporated into our Latin square algorithm. The authors would also like to thank C.J. Colbourn for helpful discussions relating to the choice of transition operation.

Appendix

In this appendix we list SFLS's of orders *16* and *18*, along with an analysis which helps display the fact that these squares are subsquare free. A similar analysis was used in [6] to show that a Latin square of side *12* was subsquare free. In this analysis we use the fact that in a Latin square L of order n any pair of rows defines a permutation of the n elements - we shall call this a row permutation. The Latin square property prescribes that there is no row permutation containing fixed elements, i.e. cycles of length 1. In addition, for L to be N_2, a necessary and sufficient condition is that there is no row permutation containing a 2-cycle. Unfortunately this doesn't generalize for L to be $p \times p$ $(2 < p \leq n)$ subsquare free (N_p). However the cycle structure of row permutations can be very useful in checking for possible subsquares of a Latin square.

We note the following points in checking for subsquares of a given N_2 Latin square L of side n:

(a) The largest possible subsquare is of side $n/2$.

(b) The existence of a 3×3 subsquare implies the existence of a set of 3 rows each pair of which has a 3-cycle in its row permutation (and on the same set of elements).

(c) Since a SFLS of order 4 does not exist, L cannot contain any subsquares of side 4 (since there are no 2×2 subsquares in L).

(d) The existence of a 5×5 subsquare implies the existence of a set of 5 rows each pair of which has a 5-cycle in its row permutation (and on the same set of elements). Note that the cycle type *(2,3)* cannot occur in any row permutation involving this square since L is N_2.

(e) Since a SFLS of order 6 does not exist, the existence of a 6×6 subsquare in L implies the existence of a 3×3 subsquare in L.

(f) In checking for possible subsquares of orders *7*, *8*, and *9*, we note that the only allowable cycle types of row permutations associated with such subsquares are as follows:

7: *(7), (4, 3)*
8: *(8), (5, 3), (4, 4)*
9: *(9), (6, 3), (5, 4), (3, 3, 3)*

The above observations lead to the following necessity check for an N_2 Latin square L of order *16* or *18* to contain subsquares. Construct a series of graphs G_i, $i = 3, 5, 7, 8$ (and *9* in the case of order *18*) in which the vertices of G_i represent the rows of L, and in which two vertices h and k are adjacent if and only if rows h and k form a permutation containing an allowable cycle type associated with i, where the allowable cycle types are as follows:

i	Allowable cycle types
3	*(3)*
5	*(5)*
7	*(7), (4, 3)*
8	*(8), (5, 3), (4, 4)*
9	*(9), (6, 3), (5, 4), (3, 3, 3)*

A clique analysis can be performed on each such graph G_i to determine that it contains no i-cliques, and therefore that the square contains no $i \times i$ subsquares.

This may not be the most efficient way to determine that L is subsquare free. However, it is useful in displaying subsquare free property of L.

As a final check we subjected each square to a dedicated subsquare checking algorithm which works as follows. Choose any pair of cells in the same row and generate the smallest square containing them. This involves $\Theta(n^2)$ amount of work for each pair. There are $\Theta(n^3)$ pairs which means that the subsquare checker has complexity $\Theta(n^5)$.

1. Subsquare free Latin square of order 16

The following is one of the SFLS's of order *16* produced by our algorithm:

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

Permutation types in above square:

```
 1:   (6,10)
 2:   (3,13)
 3:   (16)
 4:   (5,11)
 5:   (3,6,7)
 6:   (3,3,10)
 7:   (4,5,7)
 8:   (4,12)
 9:   (3,3,5,5)
10:   (3,4,9)
11:   (4,6,6)
12:   (7,9)
13:   (5,5,6)
14:   (3,5,8)
15:   (8,8)
16:   (4,4,8)
17:   (3,3,4,6)
```

Inter-row permutation cycle type structure:

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

Cycle structure clique analysis:

Clique size	Allowable cycle types	Allowable row permutation numbers	# cliques
3	(3)	2, 5, 6, 9, 10, 14, 17	8
5	(5)	4, 7, 9, 13, 14	0
7	(7), (4, 3)	5, 7, 10, 12, 17	0
8	(8), (5, 3), (4, 4)	10, 14, 15, 16, 17	0

The 8 3-cliques are:

{1,8,9}, {1,8,14}, {1,8,15}, {2,8,14}, {3,13,16}, {8,9,13}, {8,11,14}, {8,11,15}

It can be quickly confirmed that none of these cliques are associated with a 3-cycle on a common set of elements.

2. Subsquare free Latin square of order 18

The following is the SFLS of order *18* produced by our algorithm:

```
14  9 16  6 17 15  1 13  4 11 10 18  5  2  8  3 12  7
16  4  8 10  2 18  9 14 15 13 11 17  3  5  7 12  6  1
13 18  4  5  8 11  7 16 10  9 15 12 17  3  6  1 14  2
11 16  1 13 15  9 12  7 18  2  6  8  4 17 10  5  3 14
 2 14 17 12  3  1 18 11 13  6  9 10 15  7 16  4  5  8
 9  3 18  2 12  7 13  4  8  5 17  1  6 14 11 15 10 16
 5  8 14  1 11 16 17 18  9 15 12  4  2 10  3  6  7 13
18  5  7 17 14 10  8 15  2 12  1  6 16 11  4  9 13  3
10  2  3 14  6  5 11  9 16  4  8 13  7  1 18 17 15 12
 8 13 11  3  7 17  4 12  6 10 16 14  9 18 15  2  1  5
 4  7 12  8 10  2  3  6  5 17 13  9  1 16 14 11 18 15
17 15 13  9  5 14  6  1  7  3  4  2 10  8 12 18 16 11
 6 11  2 15  4 13  5 10 14  8  3  7 18 12  1 16  9 17
12 17  5  7  1  6 15  3 11 14 18 16  8 13  2 10  4  9
 7 12 10 16  9  4 14 17  3  1  5 15 11  6 13  8  2 18
15  1  9 11 18  3 10  2 12 16  7  5 13  4 17 14  8  6
 3 10  6 18 16  8  2  5  1  7 14 11 12 15  9 13 17  4
 1  6 15  4 13 12 16  8 17 18  2  3 14  9  5  7 11 10
```

Permutation types in above square:

```
 1:   (18)
 2:   (4,4,10)
 3:   (3,15)
 4:   (4,5,9)
 5:   (4,14)
 6:   (7,11)
 7:   (4,6,8)
 8:   (3,7,8)
 9:   (6,12)
10:   (5,13)
11:   (8,10)
12:   (3,5,10)
13:   (3,3,5,7)
14:   (9,9)
15:   (3,6,9)
```

16: (3,4,4,7)
17: (3,4,11)
18: (3,3,12)
19: (5,6,7)
20: (5,5,8)
21: (3,4,5,6)

Inter-row permutation cycle type structure:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1:	-	1	2	1	1	3	1	4	5	3	6	1	1	7	1	8	1	1
2:	1	-	9	10	11	1	9	3	1	12	1	11	13	10	14	11	6	10
3:	2	9	-	11	11	1	5	5	1	1	15	5	10	5	3	5	10	10
4:	1	10	11	-	16	12	3	10	1	1	8	5	10	6	14	11	11	6
5:	1	11	11	16	-	1	6	6	2	17	15	11	5	5	5	10	5	9
6:	3	1	1	12	1	-	1	1	6	3	16	12	1	4	2	1	1	4
7:	1	9	5	3	6	1	-	6	8	18	15	6	10	14	5	6	10	5
8:	4	3	5	10	6	1	6	-	15	1	1	3	5	16	13	3	9	11
9:	5	1	1	1	2	6	8	15	-	3	10	1	1	1	19	1	12	1
10:	3	12	1	1	17	3	18	1	3	-	3	12	1	1	19	20	1	1
11:	6	1	15	8	15	16	15	1	10	3	-	1	18	1	7	18	2	12
12:	1	11	5	5	11	12	6	3	1	12	1	-	5	3	5	10	9	14
13:	1	13	10	10	5	1	10	5	1	1	18	5	-	3	3	10	3	10
14:	7	10	5	6	5	4	14	16	1	1	1	3	3	-	5	3	13	11
15:	1	14	3	14	5	2	5	13	19	19	7	5	3	5	-	3	21	11
16:	8	11	5	11	10	1	6	3	1	20	18	10	10	3	3	-	6	5
17:	1	6	10	11	5	1	10	9	12	1	2	9	3	13	21	6	-	5
18:	1	10	10	6	9	4	5	11	1	1	12	14	10	11	11	5	5	-

Cycle structure clique analysis:

Clique size	Allowable cycle types	Allowable row permutation numbers	# cliques
3	(3)	3, 8, 12, 13, 15, 16, 17, 18, 21	14
5	(5)	4, 10, 12, 13, 19, 20, 21	0
7	(7), (4, 3)	6, 8, 13, 16, 17, 19, 21	0
8	(8), (5, 3), (4, 4)	2, 7, 8, 11, 12, 13, 16, 20, 21	0
9	(9), (6, 3), (5, 4), (3, 3, 3)	4, 14, 15, 21	0

The 14 3-cliques are:

{1,6, 10}, {4, 5, 11}, {4, 6, 11}, {4, 7, 11}, {5, 10, 11}, {6, 10, 11}, {6, 10, 12}, {7, 9, 10}, {7, 10, 11}, {8, 12, 14}, {8, 14, 16}, {8, 15, 16}, {13, 14, 17}, {13, 15, 17}

It can be quickly confirmed that none of these cliques are associated with a 3-cycle on a common set of elements.

References

[1] Emile Aarts, Jan Korst, *Simulated annealing and Boltzmann machines - A stochastic approach to combinatorial optimization and neural computing* (Wiley, 1989).

[2] L. D. Andersen, E. Mendelsohn, *A direct construction for Latin squares without proper subsquares*, Ann. Discrete Math. **15** (1982), 27-53.

[3] V. Cerny, *Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm*, J. Optimization Theory and Applications **45** (1985), 41-51.

[4] J. Denes, A.D. Keedwell, *Latin squares and their applications*(Academic Press, 1974).

[5] R.H.F. Denniston, *Remarks on Latin Squares with no subsquares of order two*, Utilitas Math. **13** (1978), 299-302.

[6] P.B. Gibbons, E. Mendelsohn, *The existence of a subsquare free Latin square of side 12*, Siam J. Alg. Disc. Meth. **8, 1** (1987), 93 - 99.

[7] K. Heinrich, *Latin squares with no proper subsquares*, J. Combin. Theory Ser. A (1980), 346-353.

[8] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), 671-680.

[9] A. Kotzig, C.C. Lindner, A. Rosa, *Latin squares with no subsquares of order two and disjoint Steiner triple systems*, Utilitas Math **7** (1975), 287-294.

[10] A. Kotzig, J. Turgeon, *On certain constructions for Latin squares with no subsquares of order two*, Discrete Math. **16** (1976), 263-270.

[11] P.J.M. van Laarhoven, *Theoretical and computational aspects of simulated annealing*, Erasmus University, Rotterdam, Ph.D. thesis (available as a CWI Tract, 1988).

[12] R.A. Mathon, Private communication.

[13] M. McLeish, *On the existence of Latin squares with no subsquares of order two*, Utilitas Math. **8** (1975), 41-53.

[14] E. Mendelsohn, A. Rosa, *One-factorizations of the complete graph - A survey*, J. Graph Theory **9** (1985), 43-65.

[15] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, *Equation of state calculations by fast computing machines*, J. Chem. Physics **21** (1953), 1087-1092.

[16] D.R. Stinson, *Hill-climbing algorithms for the construction of combinatorial designs*, Annals of Discrete Maths., **26** (1985), 321 - 334.