A Swap Relaxation-Based Local Search for the Latin Square Completion Problem

Zhenxuan Xie[1] , **Zhipeng Lü**[1] , **Zhouxing Su**[1*] , **Chu-Min Li**[2] , **Junwen Ding**[1] , **Yuxuan Wang**[1]
[1]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]MIS, Universite de Picardie Jules Verne, France
{zhenxuan_xie, zhipeng.lv, suzhouxing}@hust.edu.cn, chu-min.li@u-picardie.fr,
{junwending, wyx_smartlab}@hust.edu.cn

Abstract

The Latin square completion (LSC) problem aims to assign n symbols to the empty cells of a partially filled Latin square such that in each row and each column, each symbol appears exactly once. In this paper, we propose a swap relaxation-based fast local search algorithm called SRLS for solving the LSC problem. First, it introduces a novel search space definition, which forbids row conflicts based on which a swap-based neighborhood is defined. Second, a color domain relaxation technique is employed in the swap-based neighborhood by temporarily accepting the violation of some constraints to connect high-quality solutions. Third, two effective scoring functions are adopted to select neighborhood moves minimizing the number of conflicting edges as well as the number of color domain violations. Finally, SRLS employs an adaptive restart mechanism to balance the exploitation and exploration of the search. Extensive experiments on 1819 public benchmark instances demonstrate that SRLS outperforms the state-of-the-art algorithms in the literature in terms of both success rate and computational efficiency.

1 Introduction

Let n be a positive integer. A Latin square of order n is an $n \times n$ grid of cells in which each cell is filled with a symbol from 1 to n so that each symbol appears exactly once in each row and each column. One can be given a $n \times n$ grid in which some cells are already filled with a symbol but other cells are empty. The *Latin Square Completion (LSC) problem* of order n consists in filling each empty cell with a symbol to form a Latin square.

The LSC problem was first introduced by Hall [1948] and Ryser [1951]. Besides its theoretical significance as a canonical NP-complete problem [Colbourn, 1984], the LSC problem has wide applications in various domains, ranging from optical networks [Barry and Humblet, 1993], scheduling [Kumar *et al.*, 1999], error correcting codes [Colbourn *et al.*, 2004], as well as combinatorial design [Lovász *et al.*, 2003; Colbourne and Dinitz, 2007], etc.

As a challenging combinatorial problem, the LSC has attracted a lot of attention from academic society in recent decades. Among them there are several exact algorithms: Gomes and Shmoys [2002] proposed three complete solution approaches for solving the LSC problem by encoding the problem into constraint satisfaction (CSP) formulation, hybrid linear programming/CSP formulation and boolean satisfibility (SAT) formulation. In addition, Ansótegui *et al.* [2004] performed a systematic comparison of SAT and CSP models for the LSC problem.

Apart from exact algorithms, several metaheuristic algorithms have been proposed for solving the LSC problem to obtain high-quality solutions for large scale instances within reasonable time. Haraguchi [2016] presented four effective iterated local search algorithms with different neighborhoods. Jin and Hao [2019] proposed a population-based memetic algorithm called MMCOL for solving the LSC problem by transforming the $n \times n$ grid of cells to a domain-constrained Latin square graph. Pan *et al.* [2022] proposed a fast local search algorithm called FastLSC which is based on a reduction reasoning technique, a novel conflict value selection heuristic as well as a history information perturbation mechanism, after modelling the LSC problem as a graph coloring problem. To the best of our knowledge, FastLSC significantly outperforms other metaheuristic algorithms. However, there are still rooms for improvement in terms of the success rate on very hard instances.

In this paper, we also consider the LSC problem as a graph coloring problem, but directly solve it by utilizing its specific features. Specifically, we propose a swap relaxation-based local search (SRLS) algorithm, which incorporates a new search space definition, a swap relaxation-based neighborhood, two scoring functions and an adaptive restart mechanism. One of the key features of SRLS is that its search space is significantly smaller than that of MMCOL and FastLSC.

Our main contributions can be summarized as follows:

1. We propose two novel reduction rules to reduce the color domains, which can dominate the previous ones.
2. We introduce a new search space definition which imposes that each symbol appears exactly once in each row, and then this restricted search space allows us to define a

[*]Co-corresponding author.

swap-based neighborhood which exchanges the symbols of two cells on the same row and always keeps conflict-free on each row.

3. By introducing a color domain relaxation technique, we expand the search space of the swap-based neighborhood such that the search can easily connect high-quality solutions via the solutions with color domain violations. Besides, two effective scoring functions are employed to select a neighboring solution with immediate and potential impacts, which can simultaneously optimize the number of conflicting edges as well as the number of color domain violations.
4. We present a mechanism to restart the local search from the best found solution when the current solution quality is relatively poor. In addition, the threshold of judging whether the solution quality is relatively poor is adaptively updated as the local search proceeds, which balances the exploitation and exploration of the search.
5. Tested on 1819 public benchmark instances widely used in the literature, the proposed SRLS outperforms the state-of-the-art algorithms on all the tested instances in terms of both success rate and computational efficiency.
6. We conduct an ablation study to justify the merits of the key components of our SRLS algorithm, including the color domain relaxation technique, the secondary scoring function and the adaptive restart mechanism.

2 Preliminaries

A partial Latin square $\mathcal{L}_p$ of order n can be associated with a so-called Latin square graph $G = (V, E)$ [Jin and Hao, 2019], where $V = \{v_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq n\}$ is the set of all cells and v_{ij} denotes the cell on the ith row and the jth column, and E denotes the edge set where $(u, v) \in E$ iff cells u and v are on the same row or column. Clearly, $|V| = n^2$ and $|E| = n^2 (n-1)$, because there are $n(n-1)/2$ edges for each row or each column. The set of the cells on the kth row is denoted as $RV_k = \{v_{kj} \mid 1 \leq j \leq n\}$ and the set of the cells on the kth column is denoted as $CV_k = \{v_{ik} \mid 1 \leq i \leq n\}$.

The LSC problem is essentially a graph coloring problem on the associated Latin square graph $G = (V, E)$ defined above, with the symbols in Latin square being considered as the colors for G. Let $D(v)$ denote the color domain of cell v in the graph. If cell v is filled with symbol $k\,(1 \leq k \leq n)$, the color domain $D(v)$ is a single-value domain $\{k\}$, while the (initial) color domain of each unfilled cell u is $D(u) = \{1, \ldots, n\}$. The LSC problem is then to find a color mapping $C : V \to \{1, \ldots, n\}$ such that for all $v \in V$, $C(v) \in D(v)$ and for each $(u, v) \in E$, $C(v) \neq C(u)$, which is a special case of the precoloring extension problem [Biró *et al.*, 1992].

For any $(u, v) \in E$, if $C(u) = C(v)$, u and v are called conflicting cells and (u, v) is called a conflicting edge. The set of conflicting edges of solution C is represented as $CE(C)$. The quality of the solution is evaluated by $|CE(C)|$. For a conflicting edge (u, v), if u and v are on the same row, (u, v) is called a row conflict. Otherwise, (u, v) is called a column conflict. For the situation where a cell v is colored with the color not in its color domain, i.e., $C(v) \notin D(v)$, it is called a color domain violation. $DV(C)$ denotes the set of the cells which lead to color domain violations of solution C.

Based on the above notations, the LSC problem after relaxation can be formulated as follows:

$$\min\ |CE(C)|, \tag{1}$$
$$\text{s.t. } C(v) \in D(v), \forall v \in V. \tag{2}$$

Objective (1) aims to minimize the number of conflicting edges. Obviously, a feasible solution of the original LSC problem must be the optimal solution of this model, i.e., $|CE(C)| = 0$. Constraint (2) stipulates that the color of each cell must be selected from its color domain. Thus, the LSC problem is transformed from a constraint satisfaction problem to an optimization problem.

3 Swap Relaxation-Based Local Search

A Latin square is an n-coloring without any row conflict or column conflict. Therefore, if we always keep the solution without row conflicts, and change the solution only by swapping the colors of two cells on the same row, the LSC problem is simplified to only eliminating the column conflicts, so that the search space is greatly restricted.

In this study, we propose a swap relaxation-based local search algorithm called SRLS for the LSC problem, by adopting the restricted search space, which is quite different from the previous one which changes the color of one cell to another color at each iteration to eliminate both row and column conflicts. Next, we will first present the main framework of SRLS, and then describe its key components.

3.1 Main Framework

The main framework of the proposed SRLS algorithm is presented in Algorithm 1. A distinguishing feature of SRLS is that it always forbids row conflicts. Thus, the number of the cells with each color is exactly n, because each color occurs exactly once in each row, and the focus of the algorithm is on eliminating the existing column conflicts. The use of a limited search space enables the definition of a swap-based neighborhood that is tailored to exclusively address column conflicts. By narrowing the search space, a more effective and targeted solution to this issue can be devised. SRLS first applies two newly proposed reduction rules to simplify the problem instances and randomly generates an initial solution without row conflicts (line 1). Then, it iteratively improves the solution by the local search procedure (lines 3-11). At each iteration, it first evaluates the neighborhood of the current solution and selects the best move based on two scoring functions and a tabu strategy (line 4). Then, it performs the best move for the current solution (line 5). Afterwards, if the current solution C is a legal Latin square, i.e., $|CE(C)| = 0$, the current solution is returned (lines 6-7). Otherwise, it compares the quality of the current solution C with the best found solution C^* (lines 8-11). If C is not worse than C^*, i.e., $|CE(C)| \leq |CE(C^*)|$, then C^* is replaced with C (lines 8-9). Otherwise, the local search is adaptively restarted from the best found solution if necessary (lines 10-11). Finally,

Algorithm 1 The main framework of the SRLS algorithm

```
Input: A Latin square graph $G = (V, E)$
Output: A coloring solution $C$
1:  A coloring solution $C \leftarrow$ Initialize $(G)$ // Algorithm 2
2:  The best found solution $C^* \leftarrow C$
3:  while $elapsed\ time < time\ limit$ do
4:      Select the best move $m(u, v)$ based on the neighborhood evaluation and tabu strategy
5:      $C \leftarrow C \oplus m(u, v)$ // swap the colors of $u$ and $v$
6:      if $|CE(C)| = 0$ then
7:          return $C$
8:      else if $|CE(C)| \leq |CE(C^*)|$ then
9:          $C^* \leftarrow C$ // replace the best found solution
10:     else
11:         $C \leftarrow$ AdaptiveRestart $(C, C^*)$ // Algorithm 3
12: return $C^*$
```

Algorithm 2 Initialization

```
Input: A Latin square graph $G = (V, E)$, vertex set $FC$ for whose colors were fixed
Output: A coloring solution without row conflicts $C$
1:  Initialize a color domain $D(v)$ for each cell $v \in V$
2:  for $i = 1$ to $n$ do
3:      $RC_i \leftarrow \{1, \dots, n\}$ // initialize row conflict-free color
4:      $RV_i' \leftarrow RV_i$ // copy the row cells
5:  while $\exists v \in FC$ was not propagated do
6:      Propagate based on $FC$ to update color domains
7:      for $\forall v \notin FC$ and $|D(v)| = 1$ do
8:          $FC \leftarrow FC \bigcup \{v\}$
9:  for $\forall v_{ij} \in FC$ with color domain $D(v_{ij}) = k$ do
10:     $RV_i' \leftarrow RV_i' \setminus \{v_{ij}\}, RC_i \leftarrow RC_i \setminus \{k\}$ // update row cells and row conflict-free colors
11: for $i = 1$ to $n$ do
12:     for all $v \in RV_i'$ do
13:         Select a random color $c$ from $RC_i$
14:         $C(v) = c$ // assign a random row conflict-free color to the uncolored cell
15:         $RC_i \leftarrow RC_i \setminus \{c\}$
16: return $C$
```

once the elapsed time exceeds the time limit, SRLS terminates and returns the best solution C^* (line 12).

3.2 Initialization

We propose two reduction rules to reduce the color domains of some cells. Note that if the color domain of a cell is reduced to a single value, the color of the cell is fixed.

Reduction Rule 1: Let SV be a set of cells in a row (column) and $DS = \bigcup_{v \in SV} D(v)$. If $|DS| = |SV|$, then the colors in DS cannot be used to color other cells in the same row (column), and should be removed from the domains of these cells.

Reduction Rule 2: Let SV be a set of cells in a row (column) and CS be a set of colors. If $|CS| = |SV|$ and the colors in CS can only be used to color the cells in SV and cannot be used to color other cells in the same row (column), then the colors not in CS should be removed from the domain of each cell of SV.

Note that the three reduction rules proposed in MMCOL [2019] and FastLSC [2022] are the special cases of our rules for $|SV| = 1$.

Rule 1 and Rule 2 can be implemented by calling the constraint programming solver, such as Choco [Prud'homme *et al.*, 2019] with $all\text{-}different$ constraints. We construct the model with $2n$ $all\text{-}different$ constraints and repeatedly propagate the constraints until there are no more cells whose colors can be fixed. Experiments showed that our reduction rules can reduce more color domains, and can fix more cells than the previous reduction rules.

Let RC_i $(i = 1, \dots, n)$ denote the set of colors, called row conflict-free colors, such that coloring any cell on the ith row with any color in RC_i will not lead to a row conflict. The initialization procedure is shown in Algorithm 2. SRLS repeatedly applies the $all\text{-}different$ propagator to update the color domains until no cell can be colored with a fixed color (lines 5-8). After that, SRLS updates the row conflict-free colors based on the cells whose colors are fixed (lines 9-10). Then, an initial solution without row conflicts is randomly generated (lines 11-15). Specifically, for each uncolored cell u_{ij}, it is colored with a random color c from RC_i (line 14), and then RC_i is updated (line 15). Finally, an initial solution without row conflicts is returned (line 16).

3.3 Neighborhood Structure and Evaluation

In order to improve the initial solution, we employ a swap-based neighborhood structure. Denoted by $m(u, v)$, a swap move produces a neighboring solution $C \oplus m(u, v)$, by exchanging the colors of cells u and v on the same row whose colors are not fixed. For a move $m(u, v)$, u and v are called operation cells. Different from the strategies in previous literatures which always satisfy the color domain constraint, i.e., Constraint (2), we relax this constraint when performing a swap move. Specifically, for a move $m(u, v)$, even if $C(v) \notin D(u)$ (resp. $C(u) \notin D(v)$), cell u(resp. v) can still be colored with $C(v)$ (resp. $C(u)$). Note that the relaxation of the color domain is essential in our algorithm since it allows the search to connect high-quality solutions via the solutions with color domain violations. The problem with color domain relaxation can be formulated as follows:

$$\min\ |CE(C)|, \tag{3}$$

$$\text{s.t. } C(v) \in D(v), \{v \mid |D(v)| = 1\}, \tag{4}$$

$$1 \leq C(v) \leq n, \{v \mid |D(v)| \neq 1\}, \tag{5}$$

$$\bigcup_{v \in RV_i} C(v) = \{1, \dots, n\}, 1 \leq i \leq n. \tag{6}$$

Note that Constraint (6) stipulates that the row conflict is always forbidden.

The neighborhood evaluation is the most time-consuming part in a trajectory-based metaheuristic algorithm. For a typical local search algorithm which adopts the best improvement policy, it evaluates all feasible moves at each iteration, and performs the best move to optimize the scoring objectives as much as possible. If we evaluate the whole neighborhood, i.e., exchanging the colors of any two cells on the same row, there will be $O(n^3)$ swap moves, which is huge

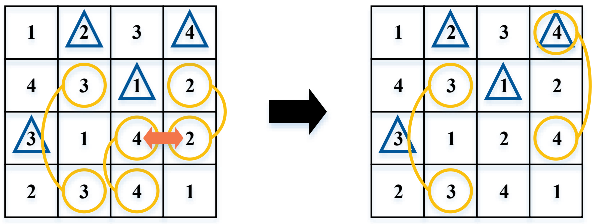


Figure 1: Illustration of the relaxation-based swap move.

for large instances. Therefore, in order to reduce the scale of the neighborhood and preserve most high-quality neighboring solutions, we only evaluate the moves where at least one operation cell is conflicting. Note that a legal solution can be obtained by only performing on the conflicting cells, because a legal Latin square is an n-coloring of the associated Latin square graph without conflicting cells. Thus, if the number of conflicting cells whose colors are not fixed in the current solution is p, the size of the neighborhood will be $O(pn)$ which is far smaller than $O(n^3)$.

Figure 1 presents an example of the relaxation-based swap move including a partial Latin square of order 4 with 4 filled cells. The numbers in cells represent the colors of the cells. The numbers in triangle denote that the colors are fixed, while the numbers in circle denote that the cells are conflicting and the edges connecting them represent the conflicting edges. One can observe that move $m(v_{33}, v_{34})$ can produce a neighboring solution which reduces one conflicting edge. Note that since the color of cell v_{14} is fixed, this move leads to a color domain violation, i.e., $C(v_{34}) \notin D(v_{34})$.

In the local search procedure, the neighborhood evaluation is an important ingredient. To obtain the best neighboring solution and improve the incumbent solution C, two scoring functions are employed by SRLS. Since the number of conflicting edges is the optimization objective, the primary scoring function is defined as minimizing the number of conflicting edges $|CE(C)|$. For a move $m(u,v)$ producing a neighboring solution C', it is evaluated as follows:

$$\Delta f_1(u,v) = |CE(C')| - |CE(C)| \tag{7}$$

In order to distinguish multiple swap moves with the same primary scoring function value, SRLS adopts a secondary scoring function which plays an important role in the search. A secondary scoring function can successfully mitigate the gradient vanishing issue and guide the search to a promising direction in many local search algorithms [Chen *et al.*, 2021; Li *et al.*, 2020; Zhang *et al.*, 2022]. For a cell v, if the color domain of cell v is violated, i.e., $C(v) \notin D(v)$, it implies that cell v is conflicting with another cell whose color is fixed, and this will lead to an illegal solution no matter what colors of the remaining cells are. For two solutions having the same number of conflicting edges, the one with fewer color domain violations is usually easier to be repaired to a legal solution. Therefore, the secondary scoring function is defined as minimizing the number of color domain violations $|DV(C)|$. Specifically, for a move $m(u,v)$ producing a neighboring solution C', it is calculated as follows:

$$\Delta f_2(u,v) = |DV(C')| - |DV(C)| \tag{8}$$

Algorithm 3 Adaptive Restart

Input: Current solution C, the best found solution C^*
Output: Input solution C for the next round of local search

```
1: Initialize at the first call: $accu \leftarrow 0, rt \leftarrow rt_0$ // initialize the restart threshold
2: if $|CE(C)| - |CE(C^*)| > rt$ then
3:     $C \leftarrow C^*$ // restart from the best found solution
       // increase the threshold $rt$ by one for every $accu_{ub}$ restarts from $C^*$ without exceeding $rt_{ub}$
4:     if $rt < rt_{ub}$ then
5:         $accu \leftarrow accu + 1$
6:         if $accu = accu_{ub}$ then
7:             $accu \leftarrow 0$ // reset
8:             $rt \leftarrow rt + 1$
9: return $C$
```

The intuitions of Δf_1 and Δf_2 lie in the fact that Δf_1 reflects the immediate impact, while Δf_2 can be considered as the potential impact. In detail, at each iteration, SRLS picks a move with the smallest Δf_1 value, and ties are broken by choosing the one with the smallest Δf_2 value. If Δf_2 values are still the same, ties are broken randomly. For more exploration, if all the neighboring solutions increase the number of conflicting edges, i.e., $\Delta f_1 > 0$, SRLS will randomly choose a conflicting cell v and select the best move associated with v based on the scoring functions. By combining these two scoring functions, SRLS can not only easily connect high-quality solutions via the solutions with color domain violations, but also minimize the number of color domain violations as soon as possible as the local search proceeds.

3.4 Tabu Search

We employ a tabu strategy in our local search. Tabu search usually incorporates a recency-based tabu list to avoid re-evaluating the just-visited solutions [Glover and Laguna, 1998]. For the tabu list, once a move is performed, the conflicting operation cell is forbidden to be recolored with the original color for the next l iterations. Specifically, for a move $m(u,v)$, if u (v) is a conflicting cell, $(u, C(u))$ $((v, C(v)))$ is recorded in the tabu list. Here, the tabu tenure l is determined by $l = \alpha * |CE(C)| + r(10)$, where $r(10)$ takes a random number in $\{1, \ldots, 10\}$ [Galinier and Hao, 1999].

A move $m(u,v)$ is declared to be tabu if coloring one cell with the color of another one is in the tabu list, i.e., $(u, C(v))$ or $(v, C(u))$ is in the tabu list. Besides, we adopt a simple aspiration criterion that a move is permitted to be selected in spite of being tabu if it leads to a solution better than the best found solution. Note that we select the best tabu move and the best non-tabu move both based on the two scoring functions mentioned above. At each iteration, the best tabu move is selected as the final move only if it is better than the best non-tabu move and it can lead to a solution whose number of conflicting edges is smaller than that of the best found solution. Otherwise, the best non-tabu move is selected.

3.5 Adaptive Restart Mechanism

Different from the classic graph coloring problem [Garey and Johnson, 1979], the symmetry of the LSC problem solution

Parameter	Description	Tested values	Value
α	Tabu tenure parameter	{0.2,0.3,0.4,0.5}	0.4
rt_0	Initial value of rt	{8,10,12}	10
rt_{ub}	Upper bound of rt	{12,15,18,21}	15
$accu_{ub}$	Updating threshold	{500,1000,1500}	1000

Table 1: Settings of important parameters.

is weak since the colors of some cells have been fixed. Thus, it is more difficult for the LSC problem to escape from a poor solution trap than the classic graph coloring problem. A restart mechanism can often help the search to break such predicaments [Lü and Hao, 2012; Wu and Hao, 2013]. Therefore, if the solution quality becomes poor enough, we employ an adaptive restart mechanism to intensify the search by restarting from the best solution found so far.

Algorithm 3 implements such an adaptive restart mechanism using three parameters rt_0, rt_{ub} and $accu_{ub}$. The key of the mechanism is a restart threshold rt. When the gap of the number of conflicting edges between the current solution C and the best solution found so far C^* exceeds rt, SRLS gives up C and restarts the local search from C^* (lines 2-3). Meanwhile, the tabu list will be cleared after restarting. For balancing the exploitation and exploration of the search, rt is initialized to a small value rt_0, and increased by one for every $accu_{ub}$ such restarts without exceeding rt_{ub} to make the search restart from C^* less often (lines 4-10).

4 Experiments and Analysis

To assess the performance of the proposed SRLS algorithm, we conduct comprehensive experiments on totally 1819 public benchmark instances. We compare the results of our algorithm with two exact solvers, Gurobi 11.0.0 and CP-SAT solver in OR-tools 9.7.2996, and six state-of-the-art heuristics, including four iterated local search algorithms (i.e., 1-ILS*, 2-ILS, 3-ILS and Tr-ILS*) [2016], MMCOL [2019] and FastLSC [2022], which are the best-performing algorithms to the best of our knowledge for the LSC problem.

4.1 Experimental Protocol

Our proposed SRLS algorithm[1] is implemented in C++ and compiled by Visual Studio 2017. The code for the reference algorithms was kindly provided by their respective authors. All experiments are carried out on Intel Xeon E5-2698v3 @ 2.30GHz CPU with 192GB RAM under Windows Server 2012 x64 and only a single thread is used per run.

There are mainly two sets of benchmark instances for the LSC problem. The first set consists of 1800 random LSC benchmark instances[2] which are classified into 18 types, each type containing 100 instances [Haraguchi, 2016]. Each type is named QWH-n-r, where n denotes the order of Latin square and r denotes the ratio of filled cells over the $n \times n$ grid of cells. The second set consists of 19 traditional benchmark instances from the COLOR03 competition[3]. For each instance, our algorithm is executed independently for 30 times with random seeds ranging from 1 to 30 under two time limits: 10 and 1000 seconds.

[1]The executable code of SRLS algorithm is online available at https://github.com/cardal1/SRLS.

[2]https://github.com/YanJINFR/Latin-Square-Completion

Instance	OR-tools		Gurobi		SRLS	
Type	$SR(\%)$	$t_{avg}\,(s)$	$SR(\%)$	$t_{avg}\,(s)$	$SR(\%)$	$t_{avg}\,(s)$
QWH-50	82.83	120.92	50.83	409.18	**99.94**	3.08
QWH-60	36.33	243.04	17.00	14.50	**100.00**	1.30
QWH-70	16.83	8.75	16.67	1.68	**100.00**	1.70
COLOR03	52.63	15.32	36.84	108.12	**92.81**	51.14

Table 2: Results of Gurobi, OR-tools and SRLS on all instances with the time limit of 1000 seconds.

The parameters of SRLS are tuned using the automatic configuration tool, irace [López-Ibáñez *et al.*, 2016]. We randomly select 300 instances as the training set which are chosen from all of the 1819 instances. The tuning process is given a budget of 9000 runs, each with a time limit of 1000 seconds. The final parameter values obtained by irace are shown in Table 1. Additionally, experiments were conducted on all possible combinations of the parameter values on the training subset, which revealed that the best parameter settings have an average of 1.02% higher success rate than other combinations of parameters. This underscores the robustness of our SRLS algorithm against parameter variations.

4.2 Computational Results

The comparison results of SRLS and the reference algorithms are presented in Tables 2-6. Note that "tl" represents the time limit. Column "$\#solv$" denotes the total number of solved runs under the time limit. Column "$SR(\%)$" displays the percentage of solved runs under the time limit. Column "$t\,(s)$" describes the mean computational time over the runs where a legal solution is obtained by the algorithm. Column "$t_{avg}\,(s)$" indicates the average value of $t\,(s)$, which is calculated over the instances in the same instance type. The numbers in bold stand for the best solution.

Table 2 displays the results obtained with a time limit of 1000 seconds by OR-tools, Gurobi and our SRLS on all instances. One can observe that our SRLS algorithm achieves over 90% success rate across all instance types, while for the state-of-the-art exact solvers, they solved much fewer instances on all the instance types, especially when n is large.

Table 3 presents the results obtained on 1800 random instances with a time limit of 10 seconds. One can observe that our SRLS algorithm achieves the highest success rate across all instance types, especially for the instances which are filled with 70% cells. Furthermore, SRLS demonstrates an average speedup of 8-106 times over the state-of-the-art algorithms. Notably, the performance of SRLS, FastLSC and MMCOL dominates that of the remaining four algorithms (i.e., 1-ILS*, 2-ILS, 3-ILS and Tr-ILS*). This phenomenon also happens under other time limits or on other benchmarks. Therefore, in the following part, we mainly compare SRLS with FastLSC and MMCOL.

[3]http://mat.gsia.cmu.edu/COLOR03/

Instance	1-ILS*		2-ILS		3-ILS		Tr-ILS*		MMCOL		FastLSC		SRLS	
Type	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$
QWH-50-30	2998	3.19	3000	0.49	2881	4.66	3000	3.44	3000	0.17	3000	0.56	3000	**0.05**
QWH-50-40	2974	2.29	3000	0.39	2739	4.59	2986	2.63	3000	0.13	3000	0.52	3000	**0.04**
QWH-50-50	2675	2.57	2987	0.66	1780	5.78	2805	3.24	3000	0.17	3000	0.53	3000	**0.03**
QWH-50-60	364	5.95	1609	3.72	43	7.68	430	6.42	2995	1.54	3000	1.19	3000	**0.05**
QWH-50-70	0	N/A	0	N/A	0	N/A	0	N/A	53	5.78	539	6.36	**2189**	3.63
QWH-50-80	3000	0.83	3000	<**0.01**	3000	0.83	3000	0.82	3000	<**0.01**	3000	<**0.01**	3000	<**0.01**
QWH-60-30	2893	7.33	3000	1.54	953	8.37	2569	7.75	3000	0.40	3000	1.02	3000	**0.13**
QWH-60-40	2929	5.11	2996	1.05	1010	7.54	2834	6.03	3000	0.32	3000	0.92	3000	**0.10**
QWH-60-50	2410	5.10	2954	1.32	395	7.66	2160	6.40	3000	0.39	3000	0.89	3000	**0.07**
QWH-60-60	100	7.19	1440	4.35	0	N/A	84	7.78	2801	3.34	2983	2.48	**3000**	0.11
QWH-60-70	0	N/A	0	N/A	0	N/A	0	N/A	7	4.25	288	7.28	**2429**	3.88
QWH-60-80	2972	1.84	2991	0.07	2971	1.93	2981	1.85	3000	<**0.01**	3000	<**0.01**	3000	<**0.01**
QWH-70-30	0	N/A	2998	3.78	0	N/A	0	N/A	3000	0.83	3000	1.75	3000	**0.29**
QWH-70-40	821	9.20	2988	2.40	36	9.40	303	9.19	3000	0.65	3000	1.49	3000	**0.21**
QWH-70-50	1391	8.42	2862	2.50	0	N/A	418	8.78	3000	0.85	3000	1.49	3000	**0.15**
QWH-70-60	17	8.56	975	5.18	0	N/A	0	N/A	1961	4.64	2846	3.88	**3000**	0.19
QWH-70-70	0	N/A	0	N/A	0	N/A	0	N/A	1	5.02	134	7.59	**2180**	4.37
QWH-70-80	2819	4.37	2935	0.53	2724	6.22	2850	4.76	3000	0.05	3000	<**0.01**	3000	<**0.01**
avg. speed-up		65.13		22.09		106.24		75.53		8.35		11.12		1.00

Table 3: Results of SRLS and other reference algorithms on 1800 random instances under the time limit of 10 seconds.

Instance	MMCOL		FastLSC		SRLS	
Type	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$	$\#solv$	$t_{avg}\,(s)$
QWH-50-60	3000	1.56	3000	1.19	3000	**0.05**
QWH-50-70	2793	243.04	2905	84.46	**2990**	18.31
QWH-60-60	3000	4.01	3000	2.53	3000	**0.11**
QWH-60-70	2930	284.40	2999	47.65	**3000**	7.36
QWH-70-60	3000	9.33	3000	4.35	3000	**0.19**
QWH-70-70	2865	386.83	3000	52.07	3000	**9.33**
avg. speed-up		35.02		14.39		1.00

Table 4: Results of MMCOL, FastLSC and SRLS on 600 random instances under the time limit of 1000 seconds.

Table 4 shows the comparison results tested under the time limit of 1000 seconds on the random instances. Instances that can be solved by all the algorithms under the 10-second limit are excluded. One can observe that SRLS solves for all runs except for 10 runs on QWH-50-70, which shows its superiority over the state-of-the-art algorithms. Furthermore, the average computational time of SRLS is significantly shorter than that of other reference algorithms.

Table 5 presents the detailed results on 13 challenging instances, out of the 1800 random instances, on which both MMCOL and FastLSC are unable to solve for all runs under the 1000-second time limit. The results indicate that SRLS is able to solve 12 out of the 13 challenging instances for all 30 runs, while for the remaining one instance (i.e., QWH-50-70-57), SRLS still outperforms MMCOL and FastLSC, with 20 runs solved compared to 4 and 8 runs, respectively.

Table 6 displays the experimental of MMCOL, FastLSC and SRLS on the 19 traditional instances with the time limits of 10s and 1000s. SRLS outperforms both MMCOL and FastLSC with higher success rates and shorter computational time on almost all these instances. In particular, for the extremely difficult instance (i.e., qwhdec.order50.holes750 [Kautz *et al.*, 2001]) where both MMCOL and FastLSC failed to solve within 1000s, SRLS achieves a success rate of 7/30, demonstrating its effectiveness and efficiency.

Instance	MMCOL		FastLSC		SRLS	
	$\#solv$	$t\,(s)$	$\#solv$	$t\,(s)$	$\#solv$	$t\,(s)$
QWH-50-70-100	9	498.17	23	296.34	**30**	50.5
QWH-50-70-13	13	495.78	21	356.11	**30**	93.05
QWH-50-70-20	6	654.84	13	373.29	**30**	160.67
QWH-50-70-21	21	398.85	28	244.98	**30**	46.42
QWH-50-70-26	6	574.38	15	423.16	**30**	229.72
QWH-50-70-57	4	671.55	8	366.92	**20**	330.63
QWH-50-70-58	13	653.35	26	323.11	**30**	40.70
QWH-50-70-70	29	347.67	28	220.32	**30**	27.31
QWH-50-70-74	17	598.83	24	417.73	**30**	112.23
QWH-50-70-8	14	582.35	25	252.79	**30**	42.19
QWH-50-70-90	22	386.87	29	190.14	**30**	27.79
QWH-50-70-98	27	403.02	28	100.08	**30**	10.96
QWH-60-70-50	16	568.99	29	250.36	**30**	51.39

Table 5: Results of MMCOL, FastLSC and SRLS on 13 challenging instances under the time limit of 1000 seconds.

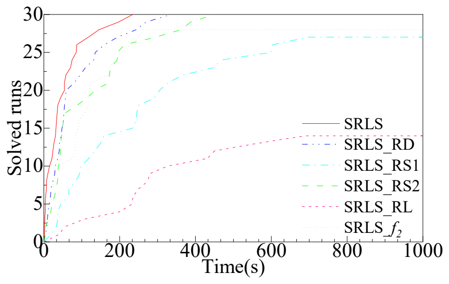
4.3 Analysis and Discussion

In order to evaluate the merits of the reduction rules, the color domain relaxation technique, the secondary scoring function and the adaptive restart mechanism employed in the SRLS algorithm, we conducted experiments on three difficult and representative instances (i.e., QWH-50-70-100, QWH-50-70-26 and QWH-60-70-50) by comparing SRLS against its alternative variants with specific components disabled.

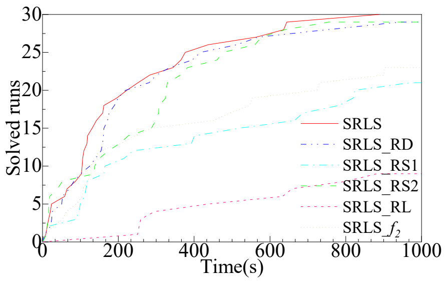
These variants include SRLS_RD (use previous reduction rules proposed in FastLSC [2022] to reduce color domains instead of those proposed in this paper and used in

Instance	*tl*=1000s						*tl*=10s					
	MMCOL		FastLSC		SRLS		MMCOL		FastLSC		SRLS	
	#*solv*	*t* (*s*)	#*solv*	*t* (*s*)	#*solv*	*t* (*s*)	#*solv*	*t* (*s*)	#*solv*	*t* (*s*)	#*solv*	*t* (*s*)
qwhdec.order18.holes120	30	<**0.01**	30	<**0.01**	30	<**0.01**	30	<**0.01**	30	<**0.01**	30	<**0.01**
qwhdec.order30.holes316	30	0.17	30	0.23	30	**0.01**	30	0.17	30	0.23	30	**0.01**
qwhdec.order30.holes320	30	0.47	30	0.36	30	**0.04**	30	0.47	30	0.36	30	**0.04**
qwhdec.order33.holes381	30	189.32	30	50.59	30	**11.39**	1	7.40	4	6.60	**18**	5.15
qwhdec.order35.holes405	30	13.16	30	5.78	30	**0.85**	17	3.97	28	4.94	**30**	0.85
qwhdec.order40.holes528	30	14.97	30	3.93	30	**0.57**	12	5.02	28	3.44	**30**	0.57
qwhdec.order5.holes10	30	<**0.01**	30	<**0.01**	30	<**0.01**	30	<**0.01**	30	<**0.01**	30	<**0.01**
qwhdec.order50.holes750	0	N/A	0	N/A	**7**	549.77	0	N/A	0	N/A	0	N/A
qwhdec.order50.holes825	30	115.25	30	20.75	30	**1.61**	0	N/A	8	7.29	**30**	1.61
qwhdec.order60.holes1080	0	N/A	2	431.40	**12**	386.60	0	N/A	0	N/A	**1**	4.78
qwhdec.order60.holes1152	29	431.11	30	65.64	30	**9.43**	0	N/A	3	8.61	**18**	4.61
qwhdec.order60.holes1440	30	2.40	30	1.96	30	**0.09**	29	2.17	30	1.96	30	**0.09**
qwhdec.order60.holes1620	30	0.82	30	1.20	30	**0.06**	30	0.82	30	1.20	30	**0.06**
qwhdec.order70.holes2450	30	0.82	30	1.51	30	**0.14**	30	0.82	30	1.51	30	**0.14**
qwhdec.order70.holes2940	30	0.64	30	1.46	30	**0.20**	30	0.64	30	1.46	30	**0.20**
qg.order100	30	10.96	30	9.98	30	**3.51**	0	N/A	17	9.26	**30**	3.51
qg.order30	30	0.03	30	0.03	30	<**0.01**	30	0.03	30	0.03	30	<**0.01**
qg.order40	30	0.14	30	0.12	30	**0.04**	30	0.14	30	0.12	30	**0.04**
qg.order60	30	0.95	30	0.80	30	**0.28**	30	0.95	30	0.80	30	**0.28**
avg. speed-up		18.84		9.31		1.00		8.78		8.44		1.00

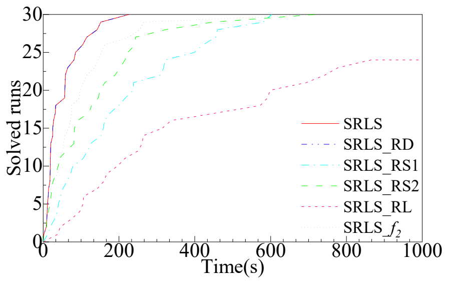
Table 6: Results of MMCOL, FastLSC and SRLS on 19 traditional instances under different time limits.



(a) QWH-50-70-100



(b) QWH-50-70-26



(c) QWH-60-70-50

Figure 2: Evolution of the solved runs of instances by SRLS and the alternative variants on three difficult instances.

SRLS), SRLS_RL (disable the color domain relaxation technique), SRLS_f_2 (deactivate the secondary scoring function), SRLS_RS1 (disable the adaptive restart mechanism) and SRLS_RS2 (restart the local search every 10^5 iterations).

Figure 2 depicts the evolution of the solved runs of instances as the search proceeds by SRLS, SRLS_RD, SRLS_RL, SRLS_f_2, SRLS_RS1 and SRLS_RS2. Each point (x, y) on the curves represents that the instance is solved for y runs within x seconds. One can observe that all these three strategies play important roles in the overall performance of SRLS. In particular, the simplified version without the color domain relaxation technique (i.e., SRLS_RL) performs the worst. The reason for this phenomenon might be that the connectivity of the solution space in SRLS_RL is poor, while the relaxation of the color domain significantly improves the connectivity of the search space. Moreover, although the simplified versions of SRLS are outperformed by SRLS, some of them are still highly competitive compared with the state-of-the-art algorithms listed in Table 5.

5 Conclusion

This paper presented a swap relaxation-based local search algorithm to solve the LSC problem. By forbidding row conflicts, we define a novel search space and an associated swap-based neighborhood. In addition, we relax the color domain to expand the search space, which enables the search to easily connect high-quality solutions. Finally, we solve the LSC problem by combining two effective scoring functions and a novel adaptive restart mechanism. Tested on 1819 public benchmark instances, SRLS outperforms the state-of-the-art algorithms in terms of both success rate and computational efficiency. Since the LSC is essentially a classical multiple permutation problem of which the structure is common in CSP [Walsh, 2001; Hnich *et al.*, 2004], we plan to adapt the techniques proposed in this paper (such as the relaxation-based neighborhood, the secondary scoring function and the adaptive restart mechanism) to solve other challenging combinatorial optimization problems in the future.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 72101094 and 62202192, and the Special Project for Knowledge Innovation of Hubei Province under Grant 2022013301015175.

References

[Ansótegui *et al.*, 2004] Carlos Ansótegui, Alvaro del Val, Iván Dotú, Cesar Fernández, and Felip Manya. Modeling choices in quasigroup completion: SAT vs. CSP. In *Proceedings of the Nineteenth AAAI Conference on Artificial Intelligence, AAAI*, pages 137–142, 2004.

[Barry and Humblet, 1993] Richard Barry and Pierre Humblet. Latin routers, design and implementation. *Journal of Lightwave Technology*, 11(5/6):891–899, May 1993.

[Biró *et al.*, 1992] Miklós Biró, Mihály Hujter, and Zs Tuza. Precoloring extension. I. Interval graphs. *Discrete Mathematics*, 100(1-3):267–279, 1992.

[Chen *et al.*, 2021] Jiejiang Chen, Shaowei Cai, Shiwei Pan, Yiyuan Wang, Qingwei Lin, Mengyu Zhao, and Minghao Yin. Nuqclq: an effective local search algorithm for maximum quasi-clique problem. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 12258–12266, 2021.

[Colbourn *et al.*, 2004] Charles Colbourn, Torleiv Klove, and Alan Ling. Permutation arrays for powerline communication and mutually orthogonal latin squares. *IEEE Transactions on Information Theory*, 50(6):1289–1291, 2004.

[Colbourn, 1984] Charles Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25–30, April 1984.

[Colbourne and Dinitz, 2007] Charles Colbourne and Jeffrey Dinitz. *Handbook of Combinatorial Designs*. CRC press Boca Raton, FL, 2007.

[Galinier and Hao, 1999] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[Garey and Johnson, 1979] Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the Theory of NP-completeness*, 1979.

[Glover and Laguna, 1998] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of Combinatorial Optimization*, pages 2093–2229. Springer, 1998.

[Gomes and Shmoys, 2002] Carla Gomes and David Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, pages 22–39, 2002.

[Hall, 1948] Marshall Hall. Distinct representatives of subsets. *Bulletin of the American Mathematical Society*, 54(10):922–926, 1948.

[Haraguchi, 2016] Kazuya Haraguchi. Iterated local search with trellis-neighborhood for the partial latin square extension problem. *Journal of Heuristics*, 22(5):727–757, October 2016.

[Hnich *et al.*, 2004] Brahim Hnich, Barbara Smith, and Toby Walsh. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research*, 21:357–391, 2004.

[Jin and Hao, 2019] Yan Jin and Jin-Kao Hao. Solving the latin square completion problem by memetic graph coloring. *IEEE Transactions on Evolutionary Computation*, 23(6):1015–1028, 2019.

[Kautz *et al.*, 2001] Henry Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla Gomes, Bart Selman, and Mark Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, pages 351–358, 2001.

[Kumar *et al.*, 1999] Ravi Kumar, Alexander Russell, and Ravi Sundaram. Approximating latin square extensions. *Algorithmica*, 24(2):128–138, 1999.

[Li *et al.*, 2020] Bohan Li, Xindi Zhang, Shaowei Cai, Jinkun Lin, Yiyuan Wang, and Christian Blum. Nucds: an efficient local search algorithm for minimum connected dominating set. In *Proceedings of the Twenty-NinthInternational Joint Conference on Artificial Intelligence, IJCAI*, pages 1503–1510, 2020.

[López-Ibáñez *et al.*, 2016] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Lovász *et al.*, 2003] László Lovász, József Pelikán, and Katalin Vesztergombi. *Discrete Mathematics: Elementary and Beyond*. Springer Science & Business Media, 2003.

[Lü and Hao, 2012] Zhipeng Lü and Jin-Kao Hao. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 218(3):865–876, 2012.

[Pan *et al.*, 2022] Shiwei Pan, Yiyuan Wang, and Minghao Yin. A fast local search algorithm for the latin square completion problem. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, pages 10327–10335, 2022.

[Prud'homme *et al.*, 2019] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. Choco solver. *Website, March*, 2019.

[Ryser, 1951] Herbert John Ryser. A combinatorial theorem with an application to latin rectangles. *Proceedings of the American Mathematical Society*, 2(4):550–552, 1951.

[Walsh, 2001] Toby Walsh. Permutation problems and channelling constraints. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 377–391. Springer, 2001.

[Wu and Hao, 2013] Qinghua Wu and Jin-Kao Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, 26:86–108, 2013.

[Zhang *et al.*, 2022] Qingyun Zhang, Zhouxing Su, Zhipeng Lü, and Lingxiao Yang. A weighting-based tabu search algorithm for the p-next center problem. In *Proceedings of the Thirty-First Joint Conference on Artificial Intelligence, IJCAI*, pages 4828–4834, 2022.