Solving the Latin Square Completion Problem by Memetic Graph Coloring

Yan Jin and Jin-Kao Hao

***Abstract*—The Latin square completion (LSC) problem involves completing a partially filled Latin square of order n by assigning numbers from 1 to n to the empty grids such that each number occurs exactly once in each row and each column. LSC has numerous applications and is, however, NP-complete. In this paper, we investigate an approach for solving LSC by converting an LSC instance to a domain-constrained Latin square graph and then solving the associated list coloring problem. To be effective, we first employ a constraint propagation-based kernelization technique to reduce the graph model and then call for a dedicated memetic algorithm to find a legal list coloring. The population-based memetic algorithm combines a problem-specific crossover operator to generate meaningful offspring solutions, an iterated tabu search procedure to improve the offspring solutions, and a distance-quality-based pool updating strategy to maintain a healthy diversity of the population. Extensive experiments on more than 1800 LSC benchmark instances in the literature show that the proposed approach can successfully solve all the instances, surpassing the state-of-the-art methods. To our knowledge, this is the first approach achieving such a performance for the considered problem. We also report computational results for the related partial Latin square extension problem.**

***Index Terms*—Graph coloring, Latin square completion (LSC), list coloring, memetic search, tabu search (TS).**

I. Introduction

A LATIN square $\mathcal{L}$ of order n is composed of $n \times n$ grids (or cells) such that each grid is filled with a number in $\{1, \ldots, n\}$ ($n \in N^+$) and each number occurs in each row and each column exactly once. If some grids of $\mathcal{L}$ remain unfilled (or empty), $\mathcal{L}$ is a partial Latin square. The Latin square completion (LSC) problem of order n involves completing the empty grids of a partial Latin square with numbers in $\{1, \ldots, n\}$ to obtain a legal Latin square.

LSC was first studied by Hall [19] and Ryser [37], and was known to be NP-complete in the general case [1], [8], [11]. LSC can be considered as a special case of the partial Latin square extension (PLSE) problem, which is to assign numbers in $\{1, \ldots, n\}$ to as many empty grids as possible under the condition that each number has to occur at most once in each row and each column. Both LSC and PLSE arise naturally in a variety of practical applications, such as scheduling, optical routing, error correcting codes as well as combinatorial design [3], [9], [16], [26], [29].

Given their theoretical and practical importance, a number of studies on LSC and PLSE have been reported in the literature. For instance, in 1999, Kumar *et al.* [26] proposed two approximation algorithms for PLSE with nontrivial worst-case performance guarantees. In 2002, Gomes and Shmoys [17] studied three complete solution methods for solving LSC: 1) a constraint satisfaction-based approach (CSP); 2) a hybrid 0/1 linear programming/CSP-based strategy (LP/CSP); and 3) a Boolean satisfiability-based approach (SAT). In 2004, Ansótegui *et al.* [2] focused on a systematic comparison of SAT and CSP models for the Latin square (quasigroup) completion problem. In 2004, Gomes *et al.* [18] presented a natural randomized rounding algorithm based on a packing linear programming relaxation, which yields an $e/(e-1)$-approximation algorithm. These algorithms are able to solve small LSC instances within a reasonable time, but fail to solve most of the large and hard instances. Recently in 2016, Haraguchi [21] introduced several powerful iterated local search algorithms with multiple neighborhoods to solve PLSE as well as LSC. Assessed on a large set of 1800 instances of various sizes and characteristics, these local search algorithms showed state-of-the-art performances. In particular, the Trellis-neighborhood search algorithm (Tr-ILS*) proves to outperform other tested ILS variants and two general optimization solvers (IBM-ILOG CPLEX and LocalSolver). The instances and the associated results presented in [21] will be used as the main references for our computational studies.

Despite much research effort dedicated to LSC and the resulting advances, there are still very few methods that are able to solve the problem effectively. For instance, no existing algorithm can find a solution for some traditional instances tested in [17] and many new instances introduced by Haraguchi [21]. It is thus quite useful and challenging to devise a method able to solve large and difficult instances.

In this paper, we investigate for the first time a solution method for solving LSC by converting the problem to a particular graph coloring problem (i.e., precoloring extension [5], then list coloring [12], [28]). With reference to the particular features of the resulting coloring model, we propose a memetic coloring algorithm (MMCOL) to solve it. Note that

Manuscript received May 23, 2018; revised October 5, 2018 and December 19, 2018; accepted February 7, 2019. Date of publication February 12, 2019; date of current version November 27, 2019. This work was supported in part by the Franco-Chinese PHC Cooperation Program under Grant 41342NC, and in part by the National Natural Science Foundation of China under Grant 61602196 and Grant 61472147. *(Corresponding author: Jin-Kao Hao.)*

Y. Jin is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: yanjin.china@hotmail.com).

J.-K. Hao is with the Department of Computer Science, LERIA, University of Angers, 49045 Angers, France, and also with the Institut Universitaire de France, 75231 Paris, France (e-mail: jin-hao.hao@univ-angers.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2019.2899053

as a heuristic, if MMCOL finds a legal coloring for a given LSC instance, then the problem is solved. Otherwise, it says nothing about whether the LSC instance is solvable or not.

We summarize the contributions of this paper as follows.

First, from a perspective of solution method, the proposed approach considers the LSC problem as a particular graph coloring problem. In this approach, we start by converting an LSC instance to a domain-constrained Latin square graph (Section II-A). Then, we reduce the graph model by applying a constraint propagation-based kernelization technique (Section II-B), leading to an instance of the list coloring problem. Finally, we seek a legal list coloring of the graph by running a dedicated memetic algorithm (Section III). The kernelization technique recursively uses constraint propagation to remove the vertices with a fixed color (corresponding to filled grids). The memetic algorithm adapts ideas from graph coloring algorithms to effectively solve the underlying list coloring problem. In particular, the algorithm integrates a problem-specific crossover to generate promising offspring solutions, an effective iterated tabu search (ITS) procedure to improve each offspring solution, and a distance-and-quality-based pool updating strategy to ensure a healthy diversity of the population.

Second, from a perspective of computational performance, we provide experimental results on a large number of LSC benchmark instances available in the literature (over 1800 in total, including 19 traditional benchmark instances from [17] and 1800 new instances from [21]) and show comparisons with various state-of-the-art approaches, including four recent iterated local search algorithms, general ILP and exact CP solvers, and a general heuristic solver. While the reference approaches can only solve a subset of the tested instances, our approach is able to solve all the instances consistently. Such a performance has never been reported in the literature, demonstrating the high effectiveness of considering LSC as a graph coloring problem and using the proposed population-based memetic algorithm to color Latin square graphs. We also adapt the method to the general PLSE problem and report computational results on additional 1800 PLSE benchmark instances from [21].

Third, and more generally, the proposed method can be used to solve the list coloring and precoloring extension problems, which are relevant graph models both in theory and in practice [28]. Indeed, for these two important coloring problems, although the literature offers many theoretical studies on specific graphs, we are not aware of any dedicated and effective algorithm able to handle large graphs. This paper thus fills in this gap. Moreover, since precoloring extension and list coloring are useful models to formulate various applications, our method can be applied in these practical settings as well.

The rest of this paper is organized as follows. Section II describes the converted graph coloring model. Section III presents the proposed MMCOL algorithm. Section IV reports computational results obtained with the proposed method and provides comparisons with state-of-the-art algorithms. Section V shows an analysis of two key components of the method, followed by concluding comments in the last section. The Appendix reports computational results of the proposed method on the related PLSE problem.

II. Latin Square Completion and Graph Coloring

A. Partial Latin Square and Latin Square Graph

Let $\mathcal{P}$ be a partial Latin square with $n \times n$ grids, an associated graph $G = (V, E)$, called Latin square graph, can be conveniently defined with the vertex set $V = \{\{1, \ldots, n\} \times \{1, \ldots, n\}\}$ ($|V| = n^2$) representing the grids and edge set E ($|E| = n^2(n-1)$) where $\{u, v\} \in E$ if and only if u and v represent two grids of the same row or column [6]. Then the LSC problem is equivalent to find a legal n-coloring of the associated Latin square graph G by using the colors $\{1, \ldots, n\}$ as follows. Let $D(v)$ denote the color domain of vertex v of the graph. Obviously, if v corresponds to a grid already filled with number k ($k \in \{1, \ldots, n\}$), $D(v)$ is a singleton domain $\{k\}$; otherwise, $D(v)$ is initially set to $\{1, \ldots, n\}$. The above coloring problem is the so-called precoloring extension problem [5], where some vertices have a fixed color and the remaining vertices are to be assigned a color in $\{1, \ldots, n\}$.

Note that a legal n-coloring of G can also be defined as a partition of V into n color classes $V_1, \ldots, V_n$ such that $\forall u, v \in V_i$ $(i = 1, \ldots, n)$, $\{u, v\} \notin E$ holds. Basically, in order to legally complete a partial Latin square, each color class must contain exactly n vertices when all the grids are filled. Let $|V_i|$ be the cardinality of color class V_i $(i = 1, \ldots, n)$, we use $|R_i| = n - |V_i|$ to denote the residual capacity of color class V_i.

Fig. 1 shows a partial Latin square of order 3, with two filled grids and seven empty grids [Fig. 1(a)] and the corresponding domain-constrained graph G with nine vertices and 18 edges [Fig. 1(b)]. Let $\mathcal{L}_{xy}$ represent the grid with xth row and yth column, then the connection between $\mathcal{L}_{xy}$ and its corresponding vertex v_i is given by $i = (x-1) \times n + y$. The objective is to find a legal three-coloring of the associated G by using the colors $\{1, 2, 3\}$. The vertices with the blue and red colors (indicated by colors 1 and 2, respectively) represent the filled grids in Fig. 1(a) while the black vertices represent the empty grids. In this example, $D(v_3) = \{1\}$ and $D(v_6) = \{2\}$ while the color domain of each uncolored vertex is $D(v_i) = \{1, 2, 3\}$ $(i = 1, 2, 4, 5, 7, 8, 9)$. The residual capacities of V_1–V_3 are 2, 2, and 3, respectively ($|R_1| = 2$, $|R_2| = 2$, and $|R_3| = 3$). Now, completing the partial Latin square is equivalent to finding a legal coloring of the graph by assigning a color in $\{1, 2, 3\}$ to each uncolored vertex of G.

One notices that unlike the general graph coloring problem, the precoloring extension problem associated to a Latin square graph has a specific feature. That is, if a vertex v of the graph represents a grid already filled with $k \in \{1, \ldots, n\}$, v has a singleton color domain $D(v) = \{k\}$ and thus receives definitively the unique color k. Moreover, this color is forbidden for any vertex u adjacent to v and should be excluded from the color domain $D(u)$. From a perspective of graph coloring, we can beneficially use this property to perform a preprocessing of the graph to obtain a reduced graph and then color the reduced graph instead of the initial Latin square graph.

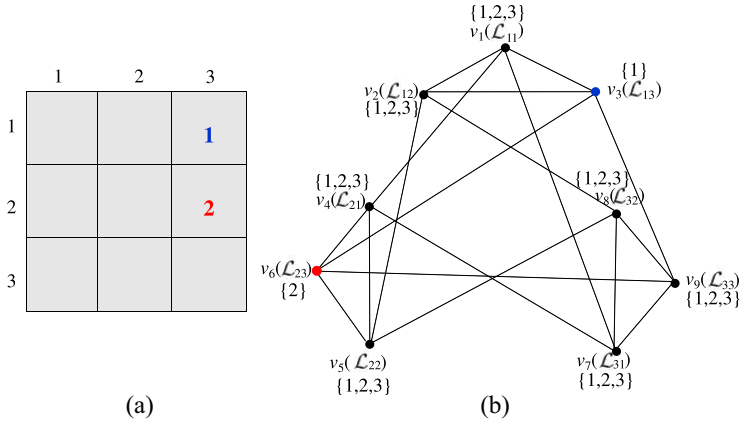

Fig. 1. Illustrative example of converting a (a) partial Latin square to a (b) domain-constrained Latin square graph.

Algorithm 1 Preprocessing Procedure for Graph Reduction

Require: A Latin square graph $G = (V, E)$ with some vertices already colored, each vertex v's color domain $D(v)$ ($v \in V$)
Ensure: A reduced graph

```
while ∃ a vertex v ∈ V with singleton color domain D(v) = {k} do
    Pick such a vertex v ∈ V with D(v) = {k} // v is colored by k
    V ← V \ {v} // Remove this colored vertex v from the graph
    E ← E \ {{u, v} ∈ E} // Remove the edges linked to v
    for each uncolored u ∈ V adjacent to v do
        D(u) ← D(u) \ {k} // Remove color k from the color domain D(u)
    end for
end while
return G = (V, E)
```

B. Preprocessing to Simplify the Latin Square Graph

The preprocessing procedure (Algorithm 1) aims to reduce the given Latin square graph by using the colored vertices (i.e., those with a singleton color domain). For this purpose, we apply a kernelization technique based on constraint propagation [36] as follows. We first remove the precolored vertices (corresponding to the filled grids) as well as the edges connected to a colored vertex. Moreover, considering the coloring constraint stating that two adjacent vertices cannot receive the same color, once a vertex v receives color k, k is forbidden for any adjacent vertex u and can be safely removed from its color domain $D(u)$. If the color domain of a vertex u becomes a singleton, vertex u definitively receives the unique color. Since vertex u is now a colored vertex, it can be used to further reduce the graph. This process is repeated until no color domain can be reduced. Notice that if the color domain of a vertex is reduced to the empty set during the preprocessing procedure, then the given LSC instance has no solution, i.e., it cannot be fully completed.

Consider again the example of Fig. 1. After applying the preprocessing to the Latin square graph in Fig. 2(a), we obtain the reduced graph shown in Fig. 2(b). In this particular case, since v_9 is connected to the two colored vertices v_3 and v_6, the colors 1 and 2 are removed from the color domain of v_9, causing $D(v_9)$ to become a singleton $\{3\}$. As a result, v_9 receives the unique color 3. The color domains of other vertices adjacent to v_3, v_6, or v_9 are also reduced, leading to the graph of Fig. 2(b) with $D(v_1) = D(v_2) = \{2, 3\}$, $D(v_4) = D(v_5) = \{1, 3\}$, and $D(v_7) = D(v_8) = \{1, 2\}$.

In terms of graph coloring, a reduced Latin square graph like Fig. 2(b) is a domain-constrained graph because the permissible colors of a vertex are limited to a list of colors in $\{1, \ldots, n\}$ [instead of the whole set $\{1, \ldots, n\}$]. In fact, the underlying coloring problem is the so-called list coloring problem [12], [28], which, like the classic vertex coloring problem, is NP-complete in the general case. Our literature review on list coloring indicates that no practical algorithm is currently able to color large graphs. Meanwhile, it is known that the list coloring problem can be transformed to the vertex coloring problem [28]. However, this transformation needs to create an auxiliary graph which is larger than the input graph by adding $k \geq n$ vertices and $\binom{k}{2}$ edges. Note that in the case of LSC, the Latin square graphs include already 2500–4900 vertices for $n = 50, 60, 70$ for the main benchmark instances tested in this paper. To our knowledge, few vertex coloring algorithms are able to effectively color graphs of these sizes given that the benchmark graphs from the well-known DIMACS Challenge (https://mat.gsia.cmu.edu/COLOR) are limited to 1000 vertices. For these reasons, we introduce below a dedicated algorithm specifically designed to solve the list coloring problem of Latin square graphs.

III. Memetic Algorithm for Coloring Latin Square Graphs

We describe in this section the population-based memetic algorithm for coloring domain-constrained Latin square graphs, i.e., solving the associated list coloring problem where each vertex v can only take a color from its given color domain $D(v)$.

A. General Procedure

The proposed algorithm (called MMCOL, shown in Algorithm 2) follows the general memetic framework which

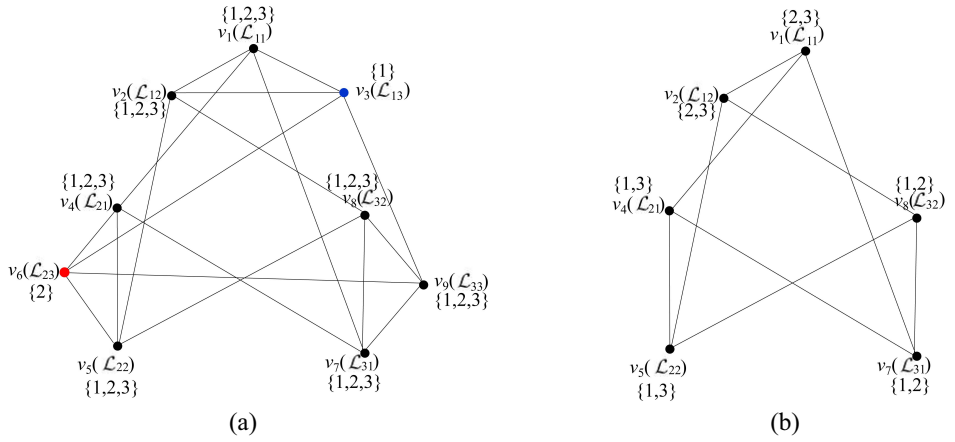

Fig. 2. Latin square graph of Fig. 1(b) and the residual Latin square graph obtained by the preprocessing procedure.

Algorithm 2 Graph Coloring Algorithm for LSC (MMCOL)

Require: A reduced Latin square graph $G = (V, E)$, the number of colors n, population size p, color domain $D(v)$ of each vertex $v \in V$

Ensure: The best n-coloring c^* and f^* found so far

1: Population_Initialization(P,p); // Generate p initial solutions of G, Sect. III-C
2: $c^* \leftarrow c$; // c^* records the best coloring found so far
3: $f^* \leftarrow f(c^*)$; // f^*records the smallest number of conflicting edges
4: **repeat**
5: $(P_1, P_2) \leftarrow$ Selection(P) // Select two parents at random for crossover
6: $o \leftarrow$ MAGX(P_1, P_2) // Crossover to get an offspring coloring, Sect. III-D
7: $o \leftarrow$ ITS(o) // Improve o with an iterated local search procedure based on Tabucol and a relaxation-based perturbation, Sect. III-E
8: **if** $f(o) < f^*$ **then**
9: $c^* \leftarrow o$; $f^* \leftarrow f(o)$;
10: **end if**
11: Population_Updating(P, o) // Use the improved offspring o to update the population, Sect. III-F
12: **until** a stopping condition is met
13: **return** f^*, c^*

combines population-based evolutionary search and local optimization [7], [13], [31], [32]. One notices that memetic approaches have proved to be highly successful to solve graph coloring and partition problems [4], [14], [24], [27], [30], [33].

The algorithm takes a reduced Latin square graph G as its input and tries to find a legal list coloring in the search space defined in Section III-B. For this purpose, the algorithm starts with an initial population [line 1 (Section III-C) in Algorithm 2]. Then, to find a legal n-coloring, MMCOL repeats a number of generations to improve the population until a stopping condition (limited to *maxGenerations*) is met. At each generation, MMCOL randomly selects two parent colorings from the population [line 5 (Section III-C) in Algorithm 2] and recombines them to generate an offspring coloring by a dedicated crossover operator [line 6 (Section III-D) in Algorithm 2]. This offspring coloring is then improved by an ITS procedure [line 7 (Section III-E) in Algorithm 2]. Finally, the improved offspring is used to update the population according to an updating strategy based on a distance-quality criterion [line 11 (Section III-F) in Algorithm 2]. During the memetic search process, if a legal coloring is found, MMCOL stops and returns the legal coloring found.

B. Search Space and Evaluation Function

Let $G = (V, E)$ be a Latin square graph with L vertices $\{v_1, \ldots, v_L\}$ and color domains $D(v_i) \subseteq \{1, \ldots, n\}$ $(i = 1, \ldots, L)$. Our MMCOL algorithm explores the following space $\mathcal{C}$ of candidate list colorings:

$$\mathcal{C} = \{c : V \rightarrow \{1, \ldots, n\} : c(v_i) \in D(v_i), i = 1, \ldots, L\}.$$

Given a candidate coloring c in $\mathcal{C}$, if $c(u) = c(v)$ and $\{u, v\} \in E$ (i.e., two adjacent vertices u and v receive the same color), then $\{u, v\}$ is a conflicting edge in c while u and v are called conflicting vertices. To assess the quality of the coloring c, we use the evaluation or fitness function f given in Equation (1), which counts the number of conflicting edges in c

$$f(c) = \sum_{\{i,j\} \in E} \max\{0, 1 - |c(v_i) - c(v_j)|\}. \quad (1)$$

Consequently, if $f(c) = 0$, c is conflict-free and identifies a legal list coloring. Otherwise ($f(c) > 0$), c is an illegal coloring with conflicting edges. For two candidate solutions c_1 and c_2, c_1 is considered to be better than c_2 if $f(c_1) < f(c_2)$ (c_1 contains fewer conflicting edges).

Given the above evaluation function, the objective of our memetic algorithm is to find a legal (conflict-free) list coloring in the search space $\mathcal{C}$ by minimizing f.

C. Population Initialization

The MMCOL algorithm applies a randomized coloring strategy to create the initial population P that is composed of p colorings sampled in $\mathcal{C}$ (p is the population size and

set to 20 in this paper). Let $G = (V, E)$ be the given graph with $V = \{v_1, \ldots, v_L\}$ and $D(v_i) \subseteq \{1, \ldots, n\}$ $(i = 1, \ldots, L)$. To build an initial coloring of G, we iteratively select an uncolored vertex v at random and then assign it a random color k from its color domain $D(v)$. Such an initial solution can be obtained very quickly in $O(L)$, but may involve a high number of conflicting edges. To obtain an initial coloring of reasonable quality, we improve this solution by the local optimization procedure (see Section III-E) and then insert the improved solution into the population if the solution does not exist in P. Otherwise, the solution is discarded and a new solution is generated. This initialization process is repeated until the population is filled up with p different colorings.

D. Crossover Operator

Recombination is an important component of our MMCOL algorithm that aims to transmit meaningful features from parents to offspring solutions [20]. For the conventional graph coloring problem, the greedy partition crossover (GPX) [14] is known to be highly effective. However, given that list coloring graphs have restricted color domains (instead of the set $\{1, \ldots, n\}$), GPX cannot be applied directly in the context of the list coloring problem. On the other hand, the key idea of GPX, i.e., inheriting large color classes, is of interest even in the case of list coloring. As a result, we propose an adaptation of GPX by taking into account the constrained color domains of our graphs. This leads to our maximum approximate group-based crossover (MAGX) for Latin square graph coloring.

The proposed MAGX crossover operator generates one offspring solution from two randomly selected parent solutions (see Algorithm 3). Let $P_1 = \{V_1^1, \ldots, V_n^1\}$ and $P_2 = \{V_1^2, \ldots, V_n^2\}$ be the parent solutions, MAGX generates, in three phases, the offspring solution $o = \{V_1^o, \ldots, V_n^o\}$, where each V_i^o $(i = 1, \ldots, n)$ is initially set to empty.

First, MAGX builds a number of color classes of o by inheriting color classes from the parent solutions. To build a new color class, MAGX selects, among the color classes of both P_1 and P_2, one largest class (call it V_i^b) such that its cardinality does not exceed the residual capacity R_i of the corresponding color class (line 3 in Algorithm 3). MAGX then uses V_i^b to form the new color class V_i^o and removes the vertices of V_i^b from both parent solutions (lines 4–6 in Algorithm 3). One notices that the color class whose cardinality is larger than its residual capacity must contain conflicting vertices. So, a color class whose cardinality is equal to (or slightly smaller than) the residual capacity is preferred in order to obtain a offspring class without conflicts. Moreover, unlike the general coloring problem where the colors are interchangeable during the recombination operation (like GPX of [14] does), for our list coloring problem of Latin square graphs, each color class of the offspring must inherit the color of the selected parent due to the constrained color domains of the vertices.

Second, for each color j such that $V_j^o = \emptyset$ in o, if V_j^1 and V_j^2 share common vertices, these vertices are used to form the color class V_j^o of the offspring.

Algorithm 3 Pseudo-Code of the MAGX Crossover Operator

Require: Parent solutions $P_1 = \{V_1^1, \ldots, V_n^1\}$, $P_2 = \{V_1^2, \ldots, V_n^2\}$, and color domain $D(v)$ of each vertex $v \in V$
Ensure: An offspring solution $o = \{V_1^o, \ldots, V_n^o\}$
1: $g \leftarrow 0$ // Count the number of color classes already built in o
2: **while** $g < n$ **do**
3: Identify from parents (P_1 and P_2) the largest color class V_i^b ($b = 1$ or 2) satisfying $|V_i^b| \leq |R_i|$ and color class V_i^o is empty
4: $V_i^o \leftarrow V_i^b$ // Color class V_i^b is transmitted to the offspring
5: Remove the vertices of V_i^b from P_1 and P_2
6: $g \leftarrow g + 1$
7: **end while**
8: **for** each empty color class V_i^o in o **do**
9: $V_i^o \leftarrow V_i^1 \bigcap V_i^2$ // For the residual vertices, transmit the vertices that share the same color in both parents
10: **end for**
11: **for** each uncolored $v \in V$ in o **do**
12: v is randomly assigned a color from its color domain $D(v)$
13: **end for**
14: **return** o

Third, for each vertex v missing in o, v is assigned a random color class in its color domain $D(v)$.

At this stage, a complete offspring solution o is obtained. In case that the offspring is the same as one of the parent solutions (this rarely happens), MAGX applies a slightly different strategy for the first phase such that the largest color class is selected by considering alternatively P_1 and P_2 (instead of considering simultaneously P_1 and P_2). Since the three phases have a time complexity of $O(n^2)$, $O(n)$, and $O(n^2)$, respectively, the time complexity of the MAGX crossover is bounded by $O(n^2)$.

Fig. 3 shows an illustration example of the MAGX crossover. This example involves a Latin square graph of order 3 ($n = 3$) with nine vertices a, b, c, d, e, f, g, h, and i to be assigned to three color classes V_1–V_3. Suppose that the color domains $D(a) = D(c) = D(g) = \{1, 3\}$, $D(h) = \{1, 2\}$, and $D(x) = \{1, 2, 3\}$ for $x \in \{b, d, e, f, i\}$. At the beginning, no color class exists in o, so $|R_i| = 3$ $(i = 1, 2, 3)$. In the first step, $V_2 = \{d, e, f\}$ of P_1 is identified as the largest color class whose $|V_2| \leq |R_2|$ and V_2 of the offspring o is empty. Thus, this color class $\{d, e, f\}$ becomes the color class V_2 of the offspring and the vertices d, e, and f are removed from both P_1 and P_2. Notice that due to the fact that vertices may have different color domains, the vertices of the inherited color class $\{d, e, f\}$ of o receives the same color as the donor parent (here color 2). Similarly, $V_3 = \{b, c, i\}$ and $V_1 = \{a, g\}$ of P_2 are inherited as color classes V_3 and V_1 of o. After these operations, vertex h is still missing in o. Since this vertex belongs to different classes in P_1 and P_2, h is assigned a random color class from its color domain $D(h) = \{1, 2\}$.

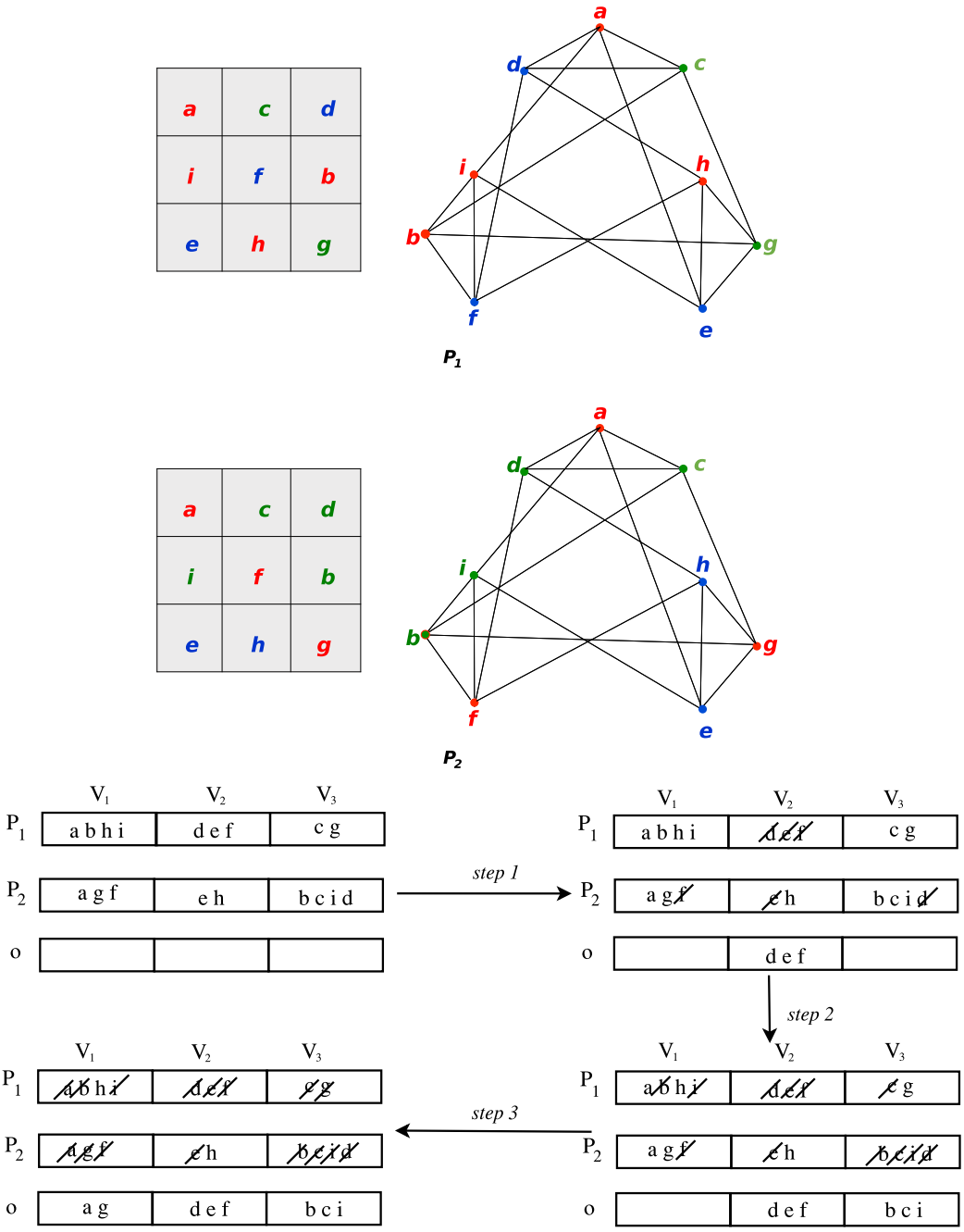

Fig. 3. Illustration of the first phase of the MAGX crossover operator (lines 2–7 in Algorithm 3).

E. Iterated Tabu Search Procedure

The ITS procedure (Algorithm 4) takes an offspring solution c generated by the MAGX crossover operator as its input and tries to improve its quality in terms of fitness function f (Equation (1), Section III-B). For this purpose, ITS iterates between a tabu search (TS) procedure followed by a relaxation-based perturbation procedure to try to attain a legal coloring by resolving the conflicts (lines 3–13 in Algorithm 4). TS iteratively improves c by recoloring conflicting vertices (see Section III-B). At the end of each TS run, if the conflicts are resolved, then a legal list coloring c^* is found, and the whole search terminates immediately. If conflicts remain in the solution, ITS triggers a perturbation procedure to modify the solution and then uses the modified solution as its starting solution for the next TS run (line 9 in Algorithm 4). ITS repeats the above process until a prefixed maximum number of iterations *maxLSIters* is reached or a legal coloring is obtained.

1) Tabu Search-Based Coloring Procedure: As its key optimization procedure, ITS uses the TS method [15] to improve a given illegal list coloring. Specifically, the TS procedure used in this paper is based on the implementations presented in [10] and [14] of the popular TabuCol algorithm for the conventional graph coloring problem [22]. Suppose that the solution c is composed of L vertices $\{v_1, v_2, \ldots, v_L\}$ and each vertex v_i receives a *permissible* color in its constrained color domain $D(v_i)$ $(i \in \{1, 2, \ldots, L\})$. The TS procedure

Algorithm 4 Pseudo-Code of ITS

Require: A n-coloring c, depth of tabu search α, color domain $D(v)$ of each vertex $v \in V$
Ensure: A legal coloring c^*

```
1: c* ← c; // c* records the best solution found so far
2: f* ← f(c*); // f* records the smallest number of conflicting edges
3: repeat
4:    (c, f) ← TS(c, α); // Apply the tabu search procedure with search depth α to improve the input coloring c, see Sect. III-E1
5:    if f < f* then
6:       c* ← c; f* ← f(c);
7:    end if
      // c is not legal coloring, trigger perturbation
8:    if f > 0 then
9:       (c, f) ← Perturbation_Procedure(c); // Apply the perturbation procedure to locate at a promising region, see Sect. III-E2
10:   else
11:      return the legal coloring c*;
12:   end if
13: until maxLSIters is reached
```

explores the space $\mathcal{C}$ composed of all possible colorings (see Section III-B) to seek a legal list coloring.

To improve the solution, TS iteratively makes transitions from the current solution c to one neighboring solution. To obtain a neighboring solution c' from solution c, TS displaces a *conflicting* vertex v from its current color class V_i to another eligible color class V_j such that $j \in D(v)$ [i.e., the current color i of vertex v is changed to a new permissible color j in v's color domain $D(v)$]. Thus, c and c' differ only by the color of a conflicting vertex v. Since the color domains are bounded by n, the size of this neighborhood is bounded by $O(n_c \times n)$, where n_c is the number of conflicting vertices in coloring c. At each iteration, TS selects among the *eligible* neighboring solutions the best neighbor c'_b according to the evaluation function f (Equation (1), Section III-B) and uses c'_b to replace c. A neighboring solution is eligible if it is not forbidden by the tabu list (see the explanation below) or if it is better than the best recorded solution. Suppose that the selected neighboring solution is obtained by changing the color i of conflicting vertex v, (v, i) is recorded in the tabu list, indicating that vertex v is forbidden to receive the color i again for the next β consecutive iterations (β is called the tabu tenure). Following [10] and [14], β is dynamically tuned by $\beta = 0.6 * f(c) + \text{random}(10)$, where random (10) is a random number in $\{1, \ldots, 10\}$. The TS procedure stops when its iteration counter reaches the given limit α (α is called the TS depth). The best solution c and the number of conflicting edges in c recorded during the search are returned as its output when the procedure terminates.

2) Relaxation-Based Perturbation: It is possible that no legal list coloring is found at the end of a TS run (see line 8 in Algorithm 4). In this case, the search is considered to be trapped in a local optimum and we trigger a relaxation-based perturbation procedure to escape from the trap.

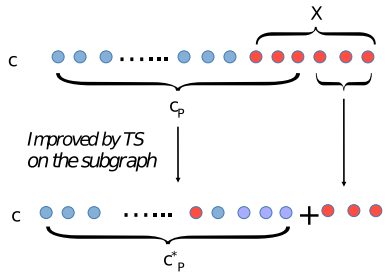

Fig. 4. Relaxation-based perturbation.

The overall procedure of the relaxation-based perturbation is illustrated in Fig. 4. Let $X \subset V$ be the set of conflicting vertices (i.e., each vertex of X is involved in at least one conflicting edge in c). The perturbation procedure is performed in three steps.

1) Extract a subgraph G' by randomly removing $\lceil |X|/2 \rceil$ conflicting vertices along with the incident edges.
2) Improve the coloring on G' using TS.
3) Construct a new coloring on G by getting it back to G.

The perturbation procedure is based on the consideration that the conflicting vertices of the local optimum are critical vertices for obtaining a legal list coloring. Meanwhile, these are also difficult vertices for conflict resolution. By ignoring some of these difficult vertices, TS has a higher chance to resolve the conflicts of the relaxed subproblem, thus providing new search opportunity when the improved solution of the relaxed subproblem is added back to the ignored partial solution. Notice that, in case that the improved c^*_p is not a legal coloring after the second step of the perturbation procedure, c^*_p is still a high quality partial coloring which could be close to a complete solution. Using c^*_p as its starting point to be extended, TS will explore a new search trajectory and hopefully encounters a legal coloring.

F. Population Updating

In order to avoid premature convergence of our MMCOL algorithm, we apply a population updating strategy similar to those used in [24], [27], [33], and [38]. The adopted strategy simultaneously considers solution quality and diversity when using an offspring solution to update the population.

Given two list colorings c_i and c_j, we use the so-called set-theoretic partition or transfer distance $D_{i,j}$ [34], [35] to measure the dissimilarity of c_i and c_j, which is defined as the minimum number of vertices that need to be moved between color classes of c_i to transform c_i to c_j. The diversity between one solution and the entire population P is given by $D_{i,P} = \min_{j \neq i}\{D_{ij}\}$. Furthermore, we define the goodness score of one n-colorings c_i of P in terms of both solution quality and diversity by $s(c_i) = f(c_i) + e^{0.08n^2/D_{i,P}}, \forall c_i \in P$ where $f(c_i)$ is the number of conflicting edges of c_i [27]. A small (large) $s(c_i)$ value indicates a good (bad) solution with respect to the individuals of P. Given offspring o, the population P is updated with o according to the following procedure.

Step 1: Insert the offspring solution o into P and compute the score $s(c_i)$ of each individual c_i of P.
Step 2: Identify the worst individual c_w (i.e., with the largest value of the scoring function s) and second

worst individual c_{sw} (with the second largest s value).

Step 3: If c_w is different from o, remove c_w from P.

Step 4: If c_w is o, remove c_w with probability 0.8 and remove c_{sw} with probability 0.2.

This updating strategy ensures that the individuals of the population are not only of high quality, but also sufficiently distanced. This property provides a basis for the random strategy used in our algorithm to select the parents for the crossover.

IV. Experimental Results

In this section, we assess the proposed approach by reporting computational results on the LSC problem and showing comparisons with state-of-the-art methods. We show in the Appendix additional results on the related PLSE problem.

A. Benchmark Instances and Experimental Protocol

To evaluate the performance of the proposed approach for solving LSC, we carry out extensive experiments on the set of 1800 random LSC benchmark instances recently introduced in [21].[1] These LSC benchmark instances (named as LSC-n-r) are evenly divided into 18 types ($n \in \{50, 60, 70\}$ and $r \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$), where n is the order of the partial Latin square and r ($r \in [0, 1]$) denotes the ratio of filled grids over the $n \times n$ grids. So each type (n, r) has 100 instances. These instances were generated by randomly removing $(1 - r)n^2$ grids from an arbitrary legal Latin square. As a result, these LSC instances always admit a complete Latin square.

By converting these instances to Latin square graphs (see Section II-A), we obtain graphs with 2500–4900 vertices and 122 500–338 100 edges.[2] To solve each instance, we first apply the preprocessing procedure of Section II-B to obtain a reduced list coloring graph which is then colored with the MMCOL algorithm. The preprocessing step takes typically from several seconds to dozens of seconds.

In addition to these 1800 random instances, we also assess our approach on the set of 19 traditional benchmark instances from the COLOR03 competition[3] that were tested, for instance, [17] and [21].

MMCOL was coded in C++[4] and compiled using g++ with the "$-O3$" option on a computer running Linux equipped with 2.83 GHz and 8-GB RAM. When running the DIMACS machine benchmark procedure "dfmax.c"[5] on our machine, we obtain the following results: 0.20, 1.23, and 4.68 s for graphs r300.5, r400.5, and r500.5, respectively. The computational results reported in this section were obtained with the parameter setting shown in Table I.

In the following sections, we first show the results on the 19 traditional instances, and then present a comparative analysis of our computational results on the large set of 1800 benchmark instances with respect to the state-of-the-art results in the literature. Given the stochastic nature of MMCOL, each instance was independently solved 30 times with different random seeds.

TABLE I
Parameter Setting

Parameter	Sect.	Description	Value
$maxLSIters$	III-A	Maximum iterations of ITS procedure	100
$maxGenerations$	III-A	Maximum number of generations	100
p	III-C	population size	20
α	III-E	Depth of tabu search	10^5

TABLE II
Computational Results on the Set of 19 Traditional Benchmark Instances

Instance			MMCOL	
Name	n	r	SR	$t(s)$
qwhdec.order5.holes10.1	5	0.60	30/30	0.00
qwhdec.order18.holes120.1	18	0.63	30/30	0.00
qg.order30	30	0.00	30/30	0.23
qwhdec.order30.holes316.1	30	0.65	30/30	0.19
qwhdec.order30.holes320.1	30	0.64	30/30	0.65
qg.order40	40	0.00	30/30	1.25
qg.order60	60	0.00	30/30	1.93
qg.order100	100	0.00	30/30	18.49
qwhdec.order35.holes405.1	35	0.67	30/30	24.80
qwhdec.order40.holes528.1	40	0.67	30/30	19.18
qwhdec.order60.holes1440.1	60	0.60	30/30	2.84
qwhdec.order60.holes1620.1	60	0.55	30/30	0.77
qwhdec.order70.holes2940.1	70	0.40	30/30	0.74
qwhdec.order70.holes2450.1	70	0.50	30/30	0.80
qwhdec.order33.holes381.bal.1	33	0.65	30/30	238.23
qwhdec.order50.holes825.bal.1	50	0.67	30/30	133.03
qwhdec.order50.holes750.bal.1	50	0.70	3/30	207.91
qwhdec.order60.holes1080.bal.1	60	0.70	5/30	397.94
qwhdec.order60.holes1152.bal.1	60	0.68	30/30	430.11

B. Results on 19 Traditional Benchmark Instances

The computational results of MMCOL on the 19 traditional Latin square graphs are summarized in Table II. Columns 1–3 of Table II indicate the characteristics of each instance: the name, the Latin square order n, and the ratio r. Columns 4 and 5 present the success rate over 30 trials (SR) and the computation time over the successful runs $t(s)$ in seconds (a successful run means that a legal Latin square is attained for this run). From Table II, one observes that MMCOL can complete the partial Latin square for all the 19 instances. Besides, our MMCOL requires a very short computation time even for the large instances with $n \geq 50$. Moreover, the last five instances are critically constrained and fully "balanced," where the number of empty grids is approximately the same over rows and columns. These instances are known to be particularly difficult [25] and only the two smallest ones of these five instances (qwhdec.order33.holes381.bal.1 and qwhdec.order50.holes825.bal.1) can be solved by very few approaches presented in [17] including CSP, hybrid strategy mixing LP/CSP, and SAT-based method. The difficulty of these instances are further confirmed by the most recent study reported in [21], where even the best performing heuristic Tr-ILS* [21] cannot solve any of these balanced instances. We also ran the source code of Tr-ILS* on our computer for a

[1] Available at http://puzzle.haraguchi-s.otaru-uc.ac.jp/PLSE/.

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

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

[4] Available at http://www.info.univ-angers.fr/hao/lsc.html.

[5] Available at ftp://dimacs.rutgers.edu/pub/dsj/clique/.

TABLE III
COMPARATIVE RESULTS OF MMCOL WITH BEST-PERFORMING ALGORITHMS ON THE SET OF 1800 LSC BENCHMARK INSTANCES

Instance			CPX-IP	CPX-CP	LSSOL	1-ILS*	2-ILS	3-ILS	Tr-ILS*	MMCOL	
n	r	$Inst\#$	$Suc\#$	$Suc\#$	$Suc\#$	$Suc\#$	$Suc\#$	$Suc\#$	$Suc\#$	$Suc\#$	$t_{avg}(s)$
50	0.3	100	9	94	10	**100**	**100**	95	**100**	**100**	0.22
	0.4	100	3	71	8	99	99	92	**100**	**100**	0.18
	0.5	100	0	12	6	96	96	83	**100**	**100**	0.22
	0.6	100	0	0	0	30	23	5	36	**100**	1.99
	0.7	100	0	0	0	0	0	0	0	**100**	299.45
	0.8	100	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	0.00
60	0.3	100	0	71	1	**100**	**100**	51	**100**	**100**	0.55
	0.4	100	0	22	0	96	99	52	**100**	**100**	0.41
	0.5	100	0	1	0	89	95	17	95	**100**	0.51
	0.6	100	0	0	0	16	12	0	23	**100**	4.77
	0.7	100	0	0	0	0	0	0	0	**100**	209.65
	0.8	100	**100**	**100**	99	98	**100**	99	99	**100**	0.00
70	0.3	100	0	34	0	**100**	**100**	19	99	**100**	1.27
	0.4	100	0	8	0	95	97	8	98	**100**	0.90
	0.5	100	0	0	0	82	87	0	84	**100**	1.03
	0.6	100	0	0	0	5	2	0	10	**100**	10.02
	0.7	100	0	0	0	0	0	0	0	**100**	272.11
	0.8	100	**100**	**100**	46	93	97	95	98	**100**	0.06
Perfect success times over 18 types (n, r)			3	3	1	4	5	1	6	18	

long computation time of 3600 s and still failed to solve any of these five balanced instances. It is thus remarkable that our MMCOL approach solves these instances consistently, even though MMCOL has a low success rate for two instances. We conclude that MMCOL performs very competitively with respect to all of the existing approaches for solving these traditional instances.

C. Comparative Results on the Set of 1800 Benchmark Instances

Table III summarizes the computational statistics of our MMCOL algorithm on the set of 1800 benchmark instances, together with the results of seven most recent methods in the literature reported in [21]. The reference methods include CPX-IP, CPX-CP, LSSOL, 1-ILS*, 2-ILS, 3-ILS, and Tr-ILS*, where CPX-IP and CPX-CP are integer programming (IP) and constraint programming (CP) solvers from IBM/ILOG CPLEX, LSSOL denotes the LocalSolver,[6] and 1-ILS*, 2-ILS, 3-ILS, and Tr-ILS* are four iterated local search algorithms with $(1, \infty)$-neighborhood, $(2, \infty)$-neighborhood, $(3, \infty)$-neighborhood, and Trellis-neighborhood search, respectively. All the reference algorithms are performed on an Intel core i7-4770 processor with 3.90 GHz and 8-GB RAM (which is faster than our computer), with a time limit of 30 s for CPX-IP, CPX-CP, and LSSOL, and 10 s for 1-ILS*, 2-ILS, 3-ILS, and Tr-ILS*. Table IV additionally presents the detailed results of our approach on a subset of 600 difficult benchmark instances.

Columns 1–3 of Table III show the characteristics of the tested instances: the order n of each Latin square, the ratio r, and the number of instances Inst# for each type (n, r). Following [21], columns 4–10 present the results of the seven reference algorithms (CPX-IP, CPX-CP, LSSOL, 1-ILS*, 2-ILS, Tr-ILS*, and 3-ILS), "*Suc#*" shows for each type of 100 instances the number of instances for which an algorithm can obtain a legal solution. Columns 11 and 12 give the results of our MMCOL algorithm in terms of "*Suc#*" and the average time $t_{\text{avg}}(s)$ in seconds is defined by $t_{\text{avg}}(s) = [(\sum_{i=1}^{i=100} t_i(s))/100]$, where $t_i(s)$ is the average time over the successful runs for the ith instance. The last row (perfect success times) shows the number of instance types with *Suc# = 100*, i.e., the number of instance types among the 18 types (n, r) for which all the 100 instances are solved by an algorithm.

From Table III, one observes that both exact methods CPX-IP and CPX-CP solve all 100 instances for only three out of the 18 types. The five ILS heuristics (LSSOL, 1-ILS*, 2-ILS, 3-ILS, and Tr-ILS*) solve all 100 instances for 1, 4, 5, 1, and 6 of 18 types (in bold), respectively. In contrast, our MMCOL algorithm can solve all the instances for all 18 types. The average time to find a solution is less than 11 s except for the 300 instances with $r = 0.7$ for which MMCOL needs less than 300 s to attain a solution while all reference algorithms fail to solve any of these instances. Besides, we observe that the seven reference algorithms have a worse performance for the types ($n \in \{50, 60, 70\}, r \in \{0.6, 0.7\}$) which are known to be more difficult [21]. On the other hand, MMCOL has no particular difficulty to solve these hard instances. In order to verify if the ILS algorithms can solve more instances by using more computation time, we ran the source code of the best performing algorithm Tr-ILS* under a much relaxed time limit of 3600 s on the instances of types ($n \in \{50, 60, 70\}, r \in \{0.6, 0.7\}$). One observes that the 100 instances of LSC-50-60 ($n = 50, r = 0.6$) can be fully completed, 98 and 95 instances of LSC-60-60 ($n = 60, r = 0.6$) and LSC-70-60 ($n = 70, r = 0.6$) can be fully completed, respectively. Nevertheless, no instance of the types ($n \in \{50, 60, 70\}, r = 0.7$) can be fully completed even if these instances have more filled grids for the LSC.

[6] http://www.localsolver.com/

TABLE IV
DETAILED COMPUTATIONAL RESULTS OF MMCOL ON A SUBSET OF 600 INSTANCES

ID	LSC-50-60		LSC-50-70		LSC-60-60		LSC-60-70		LSC-70-60		LSC-70-70	
	SR	$t(s)$	SR	$t(s)$	SR	$t(s)$	SR	$t(s)$	SR	$t(s)$	SR	$t(s)$
1	30/30	1.80	30/30	229.52	30/30	6.14	30/30	237.78	30/30	7.50	30/30	211.15
2	30/30	1.67	*29/30*	353.16	30/30	4.59	30/30	262.11	30/30	9.23	30/30	320.59
3	30/30	1.88	30/30	77.12	30/30	3.71	30/30	143.20	30/30	10.59	30/30	192.00
4	30/30	1.40	*27/30*	302.56	30/30	4.09	30/30	135.90	30/30	12.11	30/30	262.11
5	30/30	1.91	30/30	295.33	30/30	3.83	30/30	143.76	30/30	9.89	30/30	213.62
6	30/30	1.91	*29/30*	511.98	30/30	3.62	30/30	58.73	30/30	11.77	30/30	354.76
7	30/30	1.82	30/30	219.66	30/30	6.81	30/30	246.25	30/30	12.01	30/30	278.17
8	30/30	1.63	*24/30*	994.24	30/30	4.44	30/30	101.65	30/30	9.25	30/30	108.13
9	30/30	1.34	*29/30*	219.90	30/30	3.95	30/30	432.35	30/30	11.35	30/30	129.48
10	30/30	2.31	30/30	156.48	30/30	5.35	30/30	262.10	30/30	10.51	30/30	268.28
11	30/30	2.18	30/30	245.21	30/30	5.28	30/30	174.23	30/30	10.79	30/30	238.76
12	30/30	2.52	*29/30*	279.76	30/30	5.21	30/30	232.61	30/30	9.44	30/30	239.79
13	30/30	3.20	*17/30*	1154.08	30/30	4.31	30/30	178.88	30/30	9.18	30/30	310.61
14	30/30	2.47	30/30	78.89	30/30	6.62	30/30	220.65	30/30	12.60	30/30	315.94
15	30/30	1.51	30/30	55.16	30/30	4.89	30/30	277.19	30/30	8.36	30/30	392.82
16	30/30	2.28	30/30	58.69	30/30	5.02	30/30	172.64	30/30	10.69	30/30	217.64
17	30/30	2.02	30/30	315.53	30/30	6.35	30/30	171.60	30/30	10.49	30/30	280.76
18	30/30	2.17	30/30	63.03	30/30	3.54	30/30	109.55	30/30	10.52	30/30	391.01
19	30/30	2.42	30/30	140.20	30/30	4.55	30/30	212.71	30/30	10.37	30/30	146.66
20	30/30	3.23	*10/30*	912.57	30/30	4.74	30/30	197.22	30/30	12.84	30/30	260.88
21	30/30	2.46	*27/30*	516.05	30/30	4.36	30/30	124.49	30/30	10.92	30/30	185.08
22	30/30	2.26	30/30	69.85	30/30	5.08	30/30	192.56	30/30	13.49	30/30	169.89
23	30/30	2.89	30/30	170.39	30/30	4.22	30/30	189.37	30/30	10.05	30/30	411.34
24	30/30	2.45	30/30	304.91	30/30	4.82	30/30	228.04	30/30	7.06	*29/30*	472.18
25	30/30	3.16	*28/30*	598.79	30/30	5.60	30/30	396.87	30/30	9.24	30/30	711.93
26	30/30	2.35	*13/30*	1006.82	30/30	5.17	30/30	113.45	30/30	8.60	30/30	383.70
27	30/30	2.91	30/30	275.97	30/30	4.11	30/30	265.82	30/30	13.47	30/30	198.42
28	30/30	2.28	*29/30*	634.01	30/30	4.37	30/30	90.16	30/30	9.69	30/30	314.49
29	30/30	1.89	30/30	381.79	30/30	4.48	30/30	122.30	30/30	8.21	30/30	387.83
30	30/30	2.67	30/30	249.37	30/30	5.91	30/30	530.25	30/30	11.60	30/30	459.71
31	30/30	2.11	*29/30*	362.59	30/30	6.41	30/30	266.20	30/30	14.32	30/30	210.70
32	30/30	1.97	30/30	287.39	30/30	6.04	*28/30*	687.59	30/30	10.82	30/30	463.97
33	30/30	2.03	30/30	56.08	30/30	4.56	30/30	96.57	30/30	8.69	30/30	250.81
34	30/30	1.56	30/30	153.74	30/30	3.88	30/30	235.41	30/30	14.16	30/30	253.28
35	30/30	2.14	*29/30*	583.45	30/30	6.77	30/30	142.38	30/30	7.78	30/30	211.31
36	30/30	2.14	30/30	373.41	30/30	6.65	30/30	136.03	30/30	9.71	30/30	433.29
37	30/30	1.41	30/30	69.45	30/30	6.45	30/30	175.92	30/30	9.54	30/30	234.66
38	30/30	2.57	*28/30*	373.18	30/30	5.12	30/30	89.58	30/30	14.84	30/30	182.65
39	30/30	2.65	30/30	226.60	30/30	4.25	30/30	304.11	30/30	16.04	30/30	168.72
40	30/30	2.09	*29/30*	460.62	30/30	3.62	30/30	468.96	30/30	10.13	30/30	170.03
41	30/30	1.82	30/30	95.53	30/30	8.16	30/30	108.12	30/30	10.50	30/30	390.81
42	30/30	1.98	30/30	93.70	30/30	3.27	30/30	185.71	30/30	6.91	30/30	134.69
43	30/30	3.06	30/30	85.74	30/30	4.00	30/30	163.11	30/30	10.70	30/30	267.40
44	30/30	1.74	30/30	46.91	30/30	4.40	30/30	93.55	30/30	10.14	30/30	190.74
45	30/30	1.71	30/30	300.44	30/30	5.12	30/30	157.89	30/30	7.92	30/30	511.40
46	30/30	2.20	30/30	268.83	30/30	5.54	30/30	276.91	30/30	11.06	30/30	232.72
47	30/30	1.80	30/30	180.58	30/30	2.70	30/30	129.40	30/30	10.39	30/30	224.85
48	30/30	2.04	30/30	52.25	30/30	4.32	30/30	219.03	30/30	9.19	30/30	259.98
49	30/30	1.41	*29/30*	117.13	30/30	5.33	30/30	210.02	30/30	8.10	30/30	154.96
50	30/30	1.18	30/30	389.08	30/30	5.92	*27/30*	1104.26	30/30	7.59	30/30	470.39
51	30/30	2.95	*29/30*	293.51	30/30	5.87	30/30	238.48	30/30	11.52	30/30	295.06
52	30/30	1.84	30/30	71.20	30/30	5.50	30/30	261.65	30/30	10.73	30/30	167.39
53	30/30	1.50	30/30	89.39	30/30	5.09	30/30	73.60	30/30	11.03	30/30	177.52
54	30/30	1.60	30/30	137.29	30/30	4.67	30/30	108.94	30/30	10.21	30/30	145.50
55	30/30	1.29	30/30	327.82	30/30	6.11	30/30	260.51	30/30	8.25	30/30	199.61
56	30/30	1.63	30/30	137.14	30/30	4.08	30/30	141.16	30/30	9.52	30/30	383.23
57	30/30	3.63	*4/30*	1724.49	30/30	3.48	30/30	95.86	30/30	9.87	30/30	197.62
58	30/30	2.60	*23/30*	1160.19	30/30	3.39	30/30	167.31	30/30	11.81	30/30	456.38
59	30/30	2.28	30/30	276.06	30/30	7.84	30/30	213.65	30/30	7.95	30/30	677.79
60	30/30	1.43	30/30	143.88	30/30	4.50	30/30	131.84	30/30	7.89	30/30	226.17
61	30/30	1.46	30/30	27.79	30/30	5.37	30/30	144.70	30/30	6.78	30/30	259.97
62	30/30	2.52	30/30	55.34	30/30	6.00	30/30	70.70	30/30	7.85	30/30	281.82
63	30/30	1.65	30/30	193.20	30/30	4.16	30/30	219.72	30/30	10.08	30/30	298.14
64	30/30	1.61	30/30	141.12	30/30	4.02	30/30	349.99	30/30	10.41	30/30	546.86
65	30/30	1.66	30/30	293.92	30/30	4.85	30/30	130.68	30/30	6.80	30/30	156.37
66	30/30	1.63	30/30	240.08	30/30	2.92	30/30	74.45	30/30	7.06	30/30	262.80
67	30/30	2.39	30/30	95.70	30/30	6.48	30/30	157.17	30/30	8.60	30/30	198.07
68	30/30	1.28	30/30	168.85	30/30	4.84	*27/30*	781.71	30/30	6.12	30/30	474.08
69	30/30	1.23	30/30	52.04	30/30	3.77	30/30	123.35	30/30	11.13	30/30	163.11
70	30/30	2.01	*27/30*	561.12	30/30	5.19	30/30	151.39	30/30	7.00	30/30	297.63
71	30/30	1.08	30/30	106.96	30/30	4.25	30/30	168.97	30/30	10.12	30/30	194.81
72	30/30	2.91	*29/30*	423.29	30/30	3.76	30/30	106.15	30/30	11.49	30/30	456.06
73	30/30	2.58	30/30	71.39	30/30	5.78	30/30	221.44	30/30	10.78	30/30	279.10
74	30/30	2.15	*17/30*	918.90	30/30	4.84	30/30	448.23	30/30	7.58	30/30	275.67
75	30/30	1.57	30/30	202.87	30/30	4.27	30/30	189.70	30/30	5.78	30/30	192.49
76	30/30	2.18	30/30	67.06	30/30	5.68	30/30	267.95	30/30	11.53	30/30	165.14
77	30/30	1.45	30/30	55.37	30/30	6.55	30/30	159.72	30/30	11.54	30/30	189.93
78	30/30	2.30	*29/30*	265.87	30/30	5.76	30/30	197.63	30/30	12.40	30/30	162.90
79	30/30	2.45	30/30	58.50	30/30	3.56	30/30	119.92	30/30	10.48	*29/30*	478.63
80	30/30	1.33	30/30	112.48	30/30	5.11	30/30	159.74	30/30	9.22	30/30	324.70
81	30/30	1.12	30/30	124.85	30/30	4.18	30/30	233.84	30/30	11.83	30/30	365.06
82	30/30	1.56	30/30	240.88	30/30	5.28	30/30	109.26	30/30	10.49	*29/30*	318.54
83	30/30	1.26	*25/30*	438.82	30/30	3.64	30/30	61.54	30/30	14.78	30/30	305.79
84	30/30	1.28	30/30	163.42	30/30	3.98	30/30	309.11	30/30	11.62	30/30	130.06
85	30/30	1.67	30/30	44.76	30/30	4.61	30/30	120.79	30/30	8.78	30/30	122.38
86	30/30	1.91	30/30	324.36	30/30	2.99	30/30	93.56	30/30	11.39	30/30	98.84
87	30/30	1.56	30/30	518.07	30/30	3.66	30/30	184.49	30/30	9.06	30/30	140.46
88	30/30	1.15	30/30	130.92	30/30	4.47	30/30	162.81	30/30	8.56	30/30	338.45
89	30/30	1.88	30/30	497.59	30/30	4.60	30/30	176.23	30/30	8.86	30/30	233.54
90	30/30	1.65	*28/30*	968.10	30/30	3.12	30/30	171.07	30/30	10.33	30/30	261.34
91	30/30	2.94	30/30	401.46	30/30	4.36	30/30	100.40	30/30	9.65	30/30	113.15
92	30/30	2.36	30/30	118.06	30/30	4.95	30/30	366.14	30/30	8.94	30/30	127.49
93	30/30	1.22	30/30	161.35	30/30	2.66	30/30	141.83	30/30	11.57	30/30	121.07
94	30/30	1.37	30/30	297.39	30/30	3.75	30/30	95.65	30/30	10.69	30/30	218.80
95	30/30	1.93	30/30	95.87	30/30	3.21	30/30	115.31	30/30	10.56	30/30	129.73
96	30/30	1.62	30/30	175.05	30/30	4.97	30/30	386.67	30/30	6.86	30/30	251.44
97	30/30	1.93	30/30	213.69	30/30	3.53	30/30	165.69	30/30	11.01	30/30	288.06
98	30/30	1.71	*29/30*	354.62	30/30	3.89	30/30	130.48	30/30	9.58	30/30	190.03
99	30/30	1.55	30/30	204.81	30/30	4.25	30/30	136.39	30/30	6.25	30/30	222.49
100	30/30	2.73	*17/30*	550.96	30/30	5.60	30/30	292.85	30/30	9.19	30/30	404.90

Furthermore, we show in Table IV the detailed results of MMCOL on the 600 difficult instances ($n \in \{50, 60, 70\}$, $r \in \{0.6, 0.7\}$) for which most reference algorithms perform badly. For each instance, we present the success rate over 30 trials SR and the average computation time over the successful runs $t(s)$ in seconds. From Table IV, we observe that MMCOL achieves the perfect success rate 30/30 on three types ($n \in \{50, 60, 70\}$, $r = 0.6$). MMCOL has a lower success rate only for 28

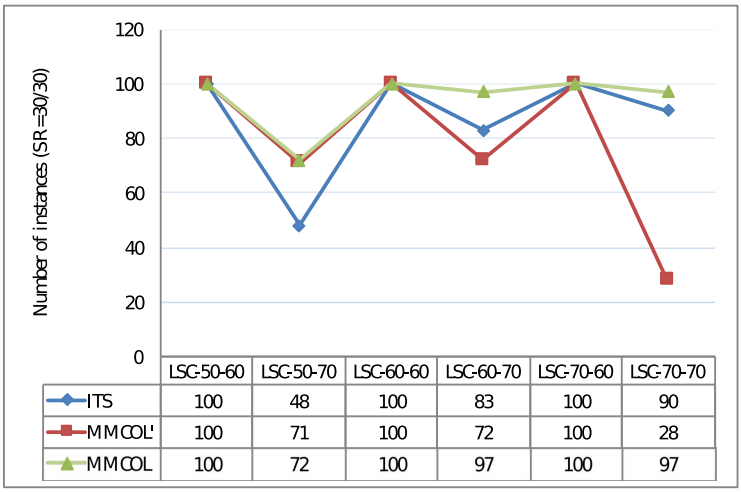

Fig. 5. Comparison of MMCOL, MMCOL', and ITS on 600 difficult instances.

instances of the type LSC-50-70 ($n = 50, r = 0.7$), three instances of the type LSC-60-70 ($n = 60, r = 0.7$), and three instances of the type LSC-70-70 ($n = 70, r = 0.7$) (in italic).

In summary, MMCOL competes very favorably with seven most recent methods in the literature and proves to be highly effective in solving the set of 1800 benchmark instances with no exception.

V. Analysis of Two Key Components

In this section, we present an analysis of two key components of the proposed method: 1) the constraint propagation-based kernelization and 2) the role of the MAGX crossover operator.

A. Impact of Constraint Propagation-Based Kernelization

The constraint propagation-based kernelization of Section II-B is used to preprocess an initial Latin square graph and thus reduces the search space of the subsequent list coloring task. In order to investigate the impact of this kernelization technique, we evaluate the reduced vertices and the color domains for the 1800 benchmark instances.

Table V summarizes the statistics of the reduced graphs for each type of instances. Columns 1–3 recall the instance characteristics: the order n, the ratio r, and the number of instances Inst#. For each type (n, r) of 100 instances, column 4 "$\#V_{\text{avg}}$" indicates the average number of the precolored vertices (i.e., the filled grids) in the initial Latin square graph. Since the color domain of some vertices becomes a singleton during the kernelization process, the graph can be further reduced. Hence, column 5 "$\#V'_{\text{avg}}$" shows the average number of further reduced vertices. Column 6 "$\#D_{\text{avg}}$" presents for each type of 100 instances the average cardinality of the color domains after kernelization, i.e., $\#D_{\text{avg}} = [(\sum_{j=1}^{j=\text{Inst\#}}(\sum_{i=1}^{i=n\times n}|D(v_i)|))/\text{Inst\#}]$. And column 7 "Solved#" gives the number of instances that are solved during the kernelization process (i.e., all color domains are reduced to a singleton).

From Table V, one observes that the vertices of the converted graphs are dramatically reduced by the kernelization technique. For the types of instances [$n \in \{50, 60, 70\}, r = 0.3, 0.4, 0.5$ and $(n = 60, r = 0.6)$], only the precolored vertices (the filled grids) are removed from the graphs, meaning that removing the precolored vertices does not lead to any new singleton domain. However, for the types of instances ($n \in \{50, 60, 70\}, r = 0.8$), many vertices can be further removed by the kernelization technique. Furthermore, the color domains of the uncolored vertices are obviously reduced from their initial sizes n. In particular, for 32 instances of the type ($n = 50, r = 0.8$) and nine instances of the type ($n = 60, r = 0.8$), a solution is found during kernelization (i.e., all color domains are reduced to a singleton), without needing to run the subsequent coloring algorithm. We conclude that the kernelization technique plays an important role in reducing the initial Latin square graphs and helps to ease the subsequent list coloring task.

TABLE V
Statistics of the Reduced Latin Square Graphs After Kernelization

Instance			$\#V_{avg}$	$\#V'_{avg}$	$\#D_{avg}$	$Solved\#$
n	r	$Inst\#$				
50	0.3	100	750.0	0.00	25.01	0
	0.4	100	1000.0	0.00	18.64	0
	0.5	100	1250.0	0.00	13.24	0
	0.6	100	1500.0	0.19	8.83	0
	0.7	100	1750.0	9.19	5.37	0
	0.8	100	2000.0	495.24	2.01	32
60	0.3	100	1080.0	0.00	29.91	0
	0.4	100	1440.0	0.00	22.23	0
	0.5	100	1800.0	0.00	15.74	0
	0.6	100	2160.0	0.00	10.43	0
	0.7	100	2520.0	4.63	6.29	0
	0.8	100	2880.0	516.63	2.43	9
70	0.3	100	1470.0	0.00	34.79	0
	0.4	100	1960.0	0.00	25.83	0
	0.5	100	2450.0	0.00	18.24	0
	0.6	100	2940.0	0.03	12.03	0
	0.7	100	3430.0	2.25	7.19	0
	0.8	100	3920.0	162.24	3.47	0

TABLE VI
COMPARATIVE RESULTS ON 22 DIFFICULT CASES OF LSC-50-70

Instance	ITS		MMCOL'		MMCOL		Instance	ITS		MMCOL'		MMCOL	
	SR	t(s)	SR	t(s)	SR	t(s)		SR	t(s)	SR	t(s)	SR	t(s)
LSC-50-70-4	24/30	988.17	**30/30**	394.21	27/30	302.56	LSC-50-70-51	26/30	643.12	27/30	627.41	**29/30**	293.51
LSC-50-70-6	24/30	585.44	28/30	722.32	**29/30**	511.98	LSC-50-70-55	26/30	495.45	**30/30**	587.49	**30/30**	327.82
LSC-50-70-8	16/30	1232.98	17/30	1664.40	**24/30**	994.23	LSC-50-70-57	1/30	1942.50	0/30	0.00	**4/30**	1724.49
LSC-50-70-13	10/30	1002.09	16/30	1375.53	**17/30**	1154.08	LSC-50-70-58	13/30	664.00	11/30	899.54	**23/30**	1160.19
LSC-50-70-20	9/30	1127.22	4/30	2086.53	**10/30**	912.57	LSC-50-70-70	21/30	716.79	25/30	1155.18	**27/30**	561.12
LSC-50-70-21	22/30	1131.82	23/30	1182.76	**27/30**	516.05	LSC-50-70-72	26/30	618.60	28/30	883.57	**29/30**	423.29
LSC-50-70-25	22/30	1029.20	**30/30**	776.40	28/30	598.79	LSC-50-70-74	8/30	1297.03	9/30	1530.22	**17/30**	918.89
LSC-50-70-26	4/30	1151.33	4/30	1166.57	**13/30**	1006.81	LSC-50-70-83	20/30	994.51	18/30	1734.15	**25/30**	438.82
LSC-50-70-28	20/30	888.14	23/30	902.15	**29/30**	634.01	LSC-50-70-87	22/30	562.83	26/30	1592.43	**30/30**	518.07
LSC-50-70-35	26/30	803.14	**30/30**	11069.10	29/30	583.45	LSC-50-70-90	20/30	1002.29	21/30	1529.61	**28/30**	968.10
LSC-50-70-40	24/30	764.63	**29/30**	893.85	**29/30**	460.62	LSC-50-70-100	14/30	1304.27	14/30	1013.30	**17/30**	550.96

B. Impact of the MAGX Crossover Operator

Within the proposed memetic algorithm, the MAGX crossover described in Section III-D is another key component. To assess its impact, we present an experiment to compare MMCOL (with the crossover MAGX), MMCOL' (with an uniform assignment crossover), and ITS (without MAGX). The uniform assignment crossover builds an offspring solution by inheriting, for each vertex, the color either from parent P_1 or P_2 with an equal probability of 0.5. For a fair comparison, we use the same parameter setting for MMCOL and MMCOL' and set $maxLSIters = 10^4$ for each ITS run, which corresponds roughly to the same search effort of MMCOL/MMCOL' with 100 generations (see Table I). The initial solutions of all the algorithms are generated by the initialization procedure given in Section III-C. We ran MMCOL, MMCOL', and ITS 30 times to solve each of the 600 difficult benchmark instances ($n \in \{50, 60, 70\}, r \in \{0.6, 0.7\}$) listed in Table IV. The comparative results are shown in Fig. 5 where we indicate for each type of 100 instances, the number of instances for which an algorithm can find a solution with a perfect success rate 30/30 (i.e., each of its 30 trials finds a legal list coloring).

From Fig. 5, one notices that both MMCOL, MMCOL', and ITS have a perfect success rate 30/30 on the 300 instances of types ($n \in \{50, 60, 70\}, r = 0.6$). On the other hand, for the other 300 instances of types ($n \in \{50, 60, 70\}, r = 0.7$), MMCOL achieves a perfect success rate 30/30 on 72, 97, and 97 instances, respectively, against 48, 83, and 90 instances for ITS and 71, 72, and 28 instances for MMCOL'.

Additionally, Table VI presents the detailed results on the 22 most difficult instances of LSC-50-70 where we show for each instance and each algorithm (MMCOL, MMCOL', and ITS), the success rate SR over 30 trials and the average computation time $t(s)$ in seconds over the successful runs. This table indicates that MMCOL obtains a higher success rate SR than MMCOL' and ITS for 17 and 22 instances, respectively (in bold), with shorter computation times $t(s)$ except for two cases. This experiment confirms that the population-based evolutionary framework implemented in our memetic algorithm and its crossover operator contribute positively to the performance of the proposed algorithm.

VI. CONCLUSION

We have proposed an approach to solve the LSC problem by converting LSC to a domain-constrained graph coloring problem. By taking advantage of the particular features of Latin square graphs, we have developed a constraint propagation-based kernelization technique to preprocess the given Latin square graph to obtain a reduced graph, for which an associated list coloring problem is defined. To effectively solve the list coloring problem, we have devised a dedicated memetic algorithm MMCOL which integrates a tailored crossover operator to generate new solutions, an ITS procedure to improve each offspring solution and a distance-quality-based pool updating strategy to ensure a healthy diversity of the population.

Extensive evaluations on a large number of benchmark instances in the literature (19 traditional instances and 1800 random instances) have shown that the proposed approach performs very well with respect to the state-of-the-art methods including those introduced very recently in 2016. In particular, our approach is able to find a solution for all the benchmark instances consistently and effectively, a performance never attained by any existing approach. We have also used a slightly modified version of the method to solve the general PLSE problem and reported computational results on the set of 2018 PLSE instances in the Appendix.

Given that LSC and PLSE have a number of applications, the proposed approach can help to solve these applications. More generally, the method proposed in this paper can be used to approximate the important list coloring and precoloring extension problems, for which few practical algorithms exist in the literature. The method will be particularly useful if large problem instances are considered.

For future work, it would be interesting to identify additional features of Latin square graphs and use them to design effective search strategies and operators. Approaches based on vertex coloring algorithms are also worthy of investigation.

APPENDIX
RESULTS ON THE PARTIAL LATIN SQUARE EXTENSION PROBLEM

We now show that the method presented in this paper can be used to solve the related and more general PLSE problem. Recall that PLSE is to assign numbers $\{1, \ldots, n\}$ to as many empty grids as possible under the condition that each number occurs at most once in each row and each column. To apply the proposed method to solve PLSE, we make the following two adjustments.

First, unlike LSC, the color domains of some vertices in the case of PLSE may become empty during the constraint propagation-based kernalization process, implying that the corresponding grids cannot be legally filled by any given number. In terms of graph coloring, if the color domain of a vertex v becomes empty when applying the preprocessing procedure, any color for vertex v is definitively conflicting with at least one of its adjacent vertices. If this happens, we randomly assign a color $\{1, \ldots, n\}$ to vertex v and keep this color unchanged during the search process.

Second, at the end of the MMCOL algorithm, there are two possibilities. If the returned final coloring c^* is conflict-free [i.e., $f(c^*) = 0$], the given partial Latin square is fully completed and an optimal solution is found. Otherwise, some vertices are assigned conflicting colors in c^* [i.e., $f(c^*) > 0$]. In this case, we obtain a legal partial solution by dropping from c^* some conflicting vertices. The dropped vertices correspond to the grids that cannot be legally filled while the remaining vertices in the legal partial coloring define a solution for the given PLSE instance. To remove conflicting vertices, we first drop any vertex v with empty color domain caused by the preprocessing procedure. Then, if conflicts remain, we repetitively remove the vertex u which is conflicting with the largest number of other vertices in the coloring until we obtain a partial conflict-free coloring.

Table VII summarizes the results of our MMCOL algorithm and seven reference methods on the set of 1800 PLSE benchmark instances introduced in [21]. Like the LSC instances of Section IV-A, these 1800 PLSE instances are evenly divided into 18 types ($n \in \{50, 60, 70\}, r \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$) [so 100 instances per type (n, r)]. For these instances, n^2 is a trivial upper bound of their optimal solutions. In this experiment, we used the same experimental condition as for solving LSC (Section IV-C). Like [21], we solved each instance once. Columns 1–3 indicate the characteristics of the instances with the same information as in Table III. Columns 4–17 present, for each reference algorithm and for each type (n, r) of 100 instances, the number of fully completed Latin squares "suc#," and the average completed grids over 100 instances "Avg." Columns 18–20 show the results of our MMCOL algorithm in terms of suc#, Avg, and the average computation time "$t_{\text{avg}}(s)$" in seconds. Notice that if a partial Latin square is fully completed, the optimum is attained (so 1186 instances out of 1800 are solved to optimality). Otherwise, the reported result in terms of the filled grids gives a lower bound of the given PLSE instance.

Table VII shows that MMCOL obtains improved or equal average results for 15 out of 18 types (in bold) except the types ($n \in \{50, 60, 70\}, r = 0.8$). In particular, for the 1000 instances of types ($n \in \{50, 60, 70\}, r \in \{0.3, 0.4, 0.5\}$), ($n = 70, r = 0.6$), and 186 instances of types ($n \in \{50, 60\}, r = 0.6$), ($n = 70, r = 0.7$), MMCOL attains an optimal solution. Meanwhile, for the instances of types ($n \in \{50, 60, 70\}, r = 0.8$), MMCOL performs worse than the reference algorithms. We mention that since the instances of types ($n \in \{50, 60, 70\}, r = 0.8$) are strongly constrained, the color domains of some uncolored vertices are reduced to the empty set during the constraint propagation-based

TABLE VII
COMPARATIVE RESULTS OF MMCOL ON THE SET OF 1800 PLSE BENCHMARK INSTANCES

Instance			CPX-IP		CPX-CP		LSSOL		1-ILS*		2-ILS		Tr-ILS*		3-ILS		MMCOL		
n	r	$Inst\#$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$Suc\#$	$Avg.$	$t_{avg}(s)$
50	0.3	100	10	2496.03	98	2499.87	13	2496.35	100	2500	99	2499.98	100	2500	98	2499.96	**100**	**2500**	0.22
	0.4	100	1	2493.78	66	2498.02	4	2494.65	99	2499.98	100	2500	100	2500	93	2499.86	**100**	**2500**	0.16
	0.5	100	0	2488.52	4	2489.92	1	2492.96	95	2499.89	98	2499.95	100	2500	67	2499.25	**100**	**2500**	0.31
	0.6	100	0	2476.21	0	2478.87	0	2489.21	7	2496.23	7	2496.3	20	2497.18	0	2494.67	**85**	**2499.64**	17.55
	0.7	100	0	2446.4	0	2451.04	0	2463.45	0	2469.47	0	2469.78	0	2470.07	0	2467.77	0	**2478.94**	3513.73
	0.8	100	0	2394.58	0	2388.1	0	2393.67	0	2394.14	0	2394.11	0	2394.14	0	2394.09	0	*2364.61*	0.10
60	0.3	100	0	3593.07	77	3598.29	0	3593.2	99	3599.98	100	3600	100	3600	64	3599.28	**100**	**3600**	0.69
	0.4	100	0	3590.68	19	3592.55	0	3591.17	99	3599.97	98	3599.96	100	3600	43	3598.58	**100**	**3600**	0.52
	0.5	100	0	3585.29	1	3585.83	0	3587.5	83	3599.65	81	3599.58	97	3599.94	21	3597.53	**100**	**3600**	0.67
	0.6	100	0	3572.61	0	3573.7	0	3585.52	5	3595.82	2	3595.85	13	3596.67	1	3592.77	**97**	**3599.94**	13.41
	0.7	100	0	3534.71	0	3540.45	0	3561.05	0	3571.47	0	3570.58	0	3572.12	0	3566.51	0	**3589.82**	4297.18
	0.8	100	0	3478.58	0	3464.14	0	3476.44	0	3478.59	0	3478.37	0	3478.49	0	3478.05	0	*3431.85*	0.10
70	0.3	100	0	4890.2	38	4893.75	0	4890.25	99	4899.98	99	4899.98	100	4900	13	4897.32	**100**	**4900**	0.90
	0.4	100	0	4887.73	5	4888.36	1	4887.98	98	4899.96	99	4899.98	99	4899.98	4	4896.4	**100**	**4900**	0.65
	0.5	100	0	4881.09	0	4881.17	0	4882.9	76	4899.41	78	4899.44	81	4899.57	0	4893.97	**100**	**4900**	1.51
	0.6	100	0	4868.21	0	4868.74	0	4877.77	2	4895.3	0	4894.93	9	4896.19	0	4888.52	**100**	**4900**	19.82
	0.7	100	0	4829.65	0	4831.94	0	4859.71	0	4872.41	0	4870.97	0	4872.95	0	4864.38	**4**	**4894.58**	4617.20
	0.8	100	0	4761.44	0	4737.73	0	4761.17	0	4766.67	0	4765.81	0	4765.91	0	4763.93	0	*4698.78*	0.10

preprocessing of Section II-B, indicating that these instances cannot be fully completed. Let $\alpha > 0$ be the number of vertices with an empty color domain identified during the preprocessing, $n^2-\alpha$ defines an upper bound of the given instance, which is strictly tighter than the trivial n^2 bound.

ACKNOWLEDGMENT

The authors would like to thank Dr. K. Haraguchi for making the benchmark instances tested in [21] available and sharing the codes of his ILS algorithm with them and Dr. U. Benlic for her comments on this paper.

REFERENCES

[1] L. Anderson, "Completing partial latin squares," *Mathematisk Fysiske Meddelelser*, vol. 41, no. 2, pp. 23–69, 1985.
[2] C. Ansótegui, A. del Val, I. Dotú, C. Fernández, and F. Manyà, "Modeling choices in quasigroup completion: SAT vs. CSP," in *Proc. AAAI*, San Jose, CA, USA, 2004, pp. 137–142.
[3] R. A. Barry and P. A. Humblet, "Latin routers, design and implementation," *J. Lightw. Technol.*, vol. 11, no. 56, pp. 891–899, May/Jun. 1993.
[4] U. Benlic and J.-K. Hao, "A multilevel memetic approach for improving graph k-partitions," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 624–642, Oct. 2011.
[5] M. Biró, M. Hujter, and Z. Tuza, "Precoloring extension. I. Interval graphs," *Discr. Math.*, vol. 100, nos. 1–3, pp. 267–279, 1992.
[6] R. C. Bose, "Strongly regular graphs, partial geometries and partially balanced designs," *Pac. J. Math.*, vol. 13, no. 2, pp. 389–419, 1963.
[7] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.
[8] C. J. Colbourn, "The complexity of completing partial latin squares," *Discr. Appl. Math.*, vol. 8, no. 1, pp. 25–30, 1984.
[9] C. J. Colbourn and J. H. Dinitz, *Handbook of Combinatorial Designs*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2006.
[10] R. Dorne and J.-K. Hao, "Tabu search for graph coloring, T-colorings and set T-colorings," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. H. Osman, and C. Roucairol, Eds. London, U.K.: Kluwer, 1998, ch. 6, pp. 77–92.
[11] T. Easton and R. G. Parker, "On completing latin squares," *Discr. Appl. Math.*, vol. 113, no. 2, pp. 167–181, 2001.
[12] P. Erdös, A. L. Rubin, and H. Taylor, "Choosability in graphs," in *Proc. West Coast Conf. Comb. Graph Theory Comput. Congr. Numerantium XXVI*, 1979, pp. 125–157.
[13] L. Feng, Y. S. Ong, M. H. Lim, and I. W. Tsang, "Memetic search with inter domain learning: A realization between CVPR and CARP," *IEEE Trans. Evol. Comput.*, vol. 19, no. 5, pp. 644–658, Oct. 2015.
[14] P. Galinier and J. K. Hao, "Hybrid evolutionary algorithms for graph coloring," *J. Comb. Optim.*, vol. 3, no. 4, pp. 379–397, 1999.
[15] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization*. Boston, MA, USA: Springer, 1998, pp. 2093–2229.
[16] V. Gogate and R. Dechter, "SampleSearch: Importance sampling in presence of determinism," *Artif. Intell.*, vol. 175, no. 2, pp. 694–729, 2011.
[17] C. Gomes and D. Shmoys, "Completing quasigroups or latin squares: A structured graph coloring problem," in *Proc. Comput. Symp. Graph Color. Generalization*, 2002, pp. 22–39.
[18] C. Gomes, R. G. Regis, and D. B. Shmoys, "An improved approximation algorithm for the partial latin square extension problem," *Oper. Res. Lett.*, vol. 32, no. 5, pp. 479–484, 2004.
[19] M. Hall, Jr., "Distinct representatives of subsets," *Bull. Amer. Math. Soc.*, vol. 54, no. 10, pp. 922–926, 1948.
[20] J.-K. Hao, "Memetic algorithms in discrete optimization," in *Handbook of Memetic Algorithms* (Studies in Computational Intelligence), vol. 379. Berlin, Germany: Springer-Verlag, 2012, ch. 6, pp. 73–94.
[21] K. Haraguchi, "Iterated local search with trellis-neighborhood for the partial latin square extension problem," *J. Heuristics*, vol. 22, no. 5, pp. 727–757, 2016.
[22] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," *Computing*, vol. 39, no. 4, pp. 345–351, 1987.
[23] T. R. Jensen and B. Toft, *Graph Coloring Problems*. New York, NY, USA: Wiley, 1995.
[24] Y. Jin and J. K. Hao, "Hybrid evolutionary search for the minimum sum coloring problem of graphs," *Inf. Sci.*, vols. 352–353, pp. 15–34, Jul. 2016.
[25] H. A. Kautz, Y. Ruan, D. Achlioptas, C. P. Gomes, and B. Selman, "Balance and filtering in structured satisfiable problems," in *Proc. 17th Int. Joint Conf. Artif. Intell. (IJCAI)*, Seattle, WA, USA, 2001, pp. 351–358.
[26] S. R. Kumar, A. Russell, and R. Sundaram, "Approximating latin square extensions," *Algorithmica*, vol. 24, no. 2, pp. 128–138, 1999.
[27] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 241–250, 2010.
[28] R. M. R. Lewis, *A Guide to Graph Colouring—Algorithms and Applications*. Cham, Switzerland: Springer Int., 2016.
[29] C. F. Laywine and G. L. Mullen, "Discrete mathematics using Latin squares," in *Series in Discrete Mathematics and Optimization*, 1st ed. New York, NY, USA: Wiley, 1998.
[30] L. Moalic and A. Gondran, "Variations on memetic algorithms for graph coloring problems," *J. Heuristics*, vol. 24, no. 1, pp. 1–24, 2018.
[31] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Norwell, MA, USA: Kluwer, 2003, pp. 105–144.
[32] F. Neri, C. Cotta, and P. Moscato, Eds., "Handbook of memetic algorithms," in *Studies in Computational Intelligence*, vol. 379. Berlin, Germany: Springer-Verlag, 2012.
[33] D. C. Porumbel, J.-K. Hao, and P. Kuntz, "An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring," *Comput. Oper. Res.*, vol. 37, no. 10, pp. 1822–1832, 2010.
[34] D. C. Porumbel, J. K. Hao, and P. Kuntz, "An efficient algorithm for computing the distance between close partitions," *Discr. Appl. Math.*, vol. 159, no. 1, pp. 53–59, 2011.
[35] S. Régnier, "Sur quelques aspects mathématiques des problèmes de classification automatique," *Mathématiques et Sci. Humaines*, vol. 82, no. 20, pp. 175–191, 1983.
[36] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of Constraint Programming*. Amsterdam, The Netherlands: Elsevier, 2006.
[37] H. J. Ryser, "A combinatorial theorem with an application to latin rectangles," *Proc. Amer. Math. Soc.*, vol. 2, no. 4, pp. 550–552, 1951.
[38] Y. Zhou, J.-K. Hao, and B. Duval, "Opposition-based memetic search for the maximum diversity problem," *IEEE Trans. Evol. Comput.*, vol. 21, no. 5, pp. 731–745, Oct. 2017.

Yan Jin received the B.Sc. and M.Sc. degrees from the Huazhong University of Science and Technology, Wuhan, China, in 2009 and 2012, respectively, and the Ph.D. degree in computer science from the University of Angers, Angers, France, in 2015.

She is currently an Assistant Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology. Her current research interests include reinforcement learning-based intelligence computing and metaheuristics for solving large-scale combinatorial optimization problems and transportation, including graph coloring problems, maximum independent set problems, packing, scheduling, and routing problems.

Jin-Kao Hao received the B.Sc. degree from the School of Computer Science, National University of Defense Technology, China, in 1982, the M.Sc. degree from the National Institute of Applied Sciences (INSA), Lyon, France, in 1987, the Ph.D. degree from the University of Franche-Comté, Besançon, France, in 1991, and the Professorship Diploma HDR (Habilitation à Diriger des Recherches) degree from the University of Science and Technology of Montpellier, Montpellier, France, in 1998.

He holds the title of Distinguished Professor (Professeur des Universités de classe exceptionnelle) of Computer Science with the University of Angers, Angers, France. He has authored or co-authored over 250 peer-reviewed publications and co-edited 9 books in Springer's LNCS series. His current research interests include design of effective algorithms and intelligent computational methods for solving large-scale combinatorial search problems, data science, complex networks, and transportation.

Dr. Hao has served as an Invited Member for over 200 program committees of international conferences and is on the Editorial Board of seven international journals. He is a Senior Fellow of the Institut Universitaire de France.