A Massively Parallel Evolutionary Algorithm for the Partial Latin Square Extension Problem

Olivier Goudet and Jin-Kao Hao *

LERIA, University of Angers, 2 Boulevard Lavoisier, 49045 Angers, France

Computers & Operations Research, 2023
https://doi.org/10.1016/j.cor.2023.106284

Abstract

The partial Latin square extension problem is to fill as many as possible empty cells of a partially filled Latin square. This problem is a useful model for a wide range of applications in diverse domains. This paper presents the first massively parallel evolutionary algorithm for this computationally challenging problem based on a transformation of the problem to partial graph coloring. The algorithm features the following original elements. Based on a very large population (with more than 10^4 individuals) and modern graphical processing units, the algorithm performs many local searches in parallel to ensure an intensive exploitation of the search space. The algorithm employs a dedicated crossover with a specific parent matching strategy to create a large number of diversified and information-preserving offspring at each generation. Extensive experiments on 1800 benchmark instances show a high competitiveness of the algorithm compared to the current best performing methods. Competitive results are also reported on the related Latin square completion problem. Analyses are performed to shed lights on the roles of the main algorithmic components. The code of the algorithm is publicly available.

Keywords: Combinatorial optimization, evolutionary search, parallel search, heuristics, partial graph coloring, Latin square problems.

* Corresponding author.
Email addresses: `olivier.goudet@univ-angers.fr` (Olivier Goudet), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao).

1 Introduction

Given a $n \times n$ grid and n distinct symbols, a Latin square $\mathcal{L}$ of order n is the grid filled with these n symbols such that each symbol appears exactly once in each row and each column (Latin square condition). A partial Latin square of order n verifies that some cells of the grid are pre-filled such that each symbol appears at most once in each row and each column. Given a partial Latin square, the partial Latin Square Extension Problem (PLSE) is to fill as many empty cells as possible. The Latin Square Completion Problem (LSC) (also known as the Quasi-Group Completion Problem) is the decision version that determines whether it is possible to fill the remaining empty cells in a given partial Latin square. Figure 1 shows an instance of PLSE with $n = 3$ where the symbols are integer numbers and the red numbers correspond to the filled cells. Two different optimal solutions to this PLSE instance with a score of 7 are shown (it is impossible to complete the grid).

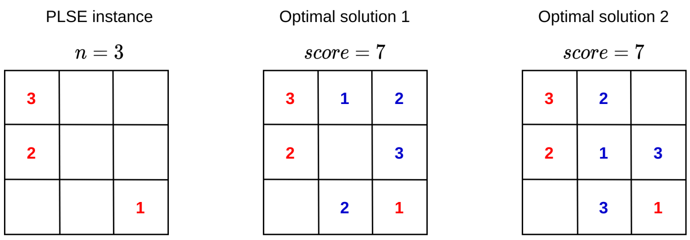

Fig. 1. Example of a PLSE instance of order $n = 3$ with 3 pre-filled cells in red. This instance has an optimal score of 7 corresponding to the maximal number of cells that can be filled without violating the Latin square condition.

Latin square problems naturally appear in numerous applications, such as scheduling, error correcting codes, as well as experimental and combinatorial design [1,2]. For instance, a typical application of the PLSE is the design of optical router systems [3].

The LSC is known to be NP-complete [4]. As the result, both the decision problem (LCS) and the optimization problem (PLSE) are computationally challenging in the general case. Due to their importance, Latin square problems have been studied from a wide variety of perspectives in different fields.

In algebra, the multiplication table of a finite quasigroup corresponds to a Latin square [5]. As such, Latin squares have been studied as a mathematical object and various properties were established [6–8].

The LSC can be expressed as an integer program with n^3 Boolean variables $x_{i,j,k}$, where $x_{i,j,k} = 1$ indicates that the cell in position (i, j) receives the symbol $k \in \{1, \ldots, n\}$. With this formulation and using integer programming solvers, optimal results were reported for small instances in [9]. The authors also investigated two other exact methods based on constraint programming

(CP) and SAT technologies. A systematic comparison of SAT and CP models was presented in [10]. An approximation algorithm was proposed based on a packing integer programming formulation in [11].

In terms of practical solving of the PLSE, a notable work was presented by Haraguchi [12]. In that work, a partial Latin square was represented using an orthogonal array, with a set of triples in $[n]^3$, such that each element (v_1, v_2, v_3) in this set indicates that the symbol v_3 is assigned to (v_1, v_2). If the Hamming distance between each pair of triples in this set is at least two, this set corresponds to a partial Latin square. Based on this representation, the author proposed several iterated local search algorithms that aim to extend the current set of triples without adding conflicts. To evaluate the practical performance of these iterated local search algorithms, the author introduced a set of 1800 instances for PLSE and another set of 1800 instances for LSC with various features (see Section 4.1 and Appendix B). The computational results showed that the iterated local search algorithms perform extremely well and outperform previous methods including integer programming, constraint programming as well as their hybrid approach.

The problem of extending a partial Latin square can also be studied from the perspective of (partial) graph coloring [13]. Indeed, a Latin square of order n can be mapped to a graph such that each vertex corresponds to a cell of the grid (there are thus n^2 vertices), and an edge exists between two vertices corresponding to two cells of the same row or column (there are thus $n^2(n-1)$ edges). The vertex of a cell pre-filled with a symbol k receives the color $k \in \{1, \ldots, n\}$. Empty cells are not colored. The PLSE consists in coloring as many uncolored vertices as possible so that two adjacent vertices do not share the same color. Based on this observation, Jin and Hao [14] proposed in 2019 a powerful memetic algorithm (MMCOL) for the related Latin square completion problem (LCS) and solved the 1800 LSC instances introduced in [12] as well as the 19 traditional LSC instances in the literature [9]. With some slight adaptations to their algorithm, they also reported excellent results on the 1800 PLSE instances of [12]. Very recently (2022), Pan et al. [15] presented a fast local search algorithm for the related LSC, which improved the solution time for most LSC instances in the literature, but didn't report results for the PLSE.

To sum, the three most recent studies on the PLSE [12] and the related LSC [14,15] significantly contributed to the practical solving of these two challenging problems. In particular, all the existing LSC benchmark instances have been solved by the MMCOL algorithm [14] and the recent FastLSC algorithm [15]. On the contrary, this is not the case for the PLSE and there is still room for improvement in terms of better solving the PLSE instances. In fact, for almost half of the 1800 benchmark instances, their optimal solutions are still unknown and only lower bounds were reported.

Motivated by this observation, this work aims to advance the state-of-the-art of solving the PLSE by establishing record-breaking lower bounds for the unsolved PLSE instances. For this purpose, we introduce the first massively parallel evolutionary algorithm for this problem that fully takes advantage of the GPU architecture to parallelize all critical search components. We summarize the contributions of the work presented in this paper as follows.

From the perspective of algorithm design, the proposed algorithm relies on a very large population P ($|P| > 10^4$) that enables massively parallel local optimization and offspring generation on the GPU architecture. This is in sharp contrast to the typical use of a small population P (typically $|P| < 10^2$) and sequential computations of many memetic algorithms including the MM-COL algorithm (e.g., [16,17,14]). The algorithm features several complementary and original search components including a parametrized asymmetric uniform crossover and an effective local search. The crossover uses a probability to control the inherited information from the parents according to a distance metric and a specific parent matching strategy to create a large number of diversified and information-preserving offspring. The local search utilizes a two-phase approach to effectively explore an enlarged search space. The algorithm is further reinforced by a parallel distance calculation procedure that enables a fast population updating.

From the perspective of computational performance, we demonstrate a high competitiveness of our algorithm on the 1800 PLSE benchmark instances from [12]. We report many improved best lower bounds for large and difficult instances, including 25 record optimal solutions. We also test the algorithm on the related LSC and show that the algorithm is able to solve all the existing benchmark instances as well (1800 from [12] and 19 from [9]).

Finally, we contribute to the understanding of the population size, the crossover and the parent matching strategy for a large population. In particular, we show that the random parent matching strategy which is typically employed in many memetic algorithms (e.g., [18,14]) is no more suitable in the context of a large population and can be beneficially replaced by a neighborhood matching strategy for a better efficiency.

In the rest of the paper, we present the solution approach and the proposed algorithm (Sections 2 and 3), experimental results and comparisons with the state-of-the-art methods (Section 4), followed by analyses of key algorithmic components and conclusions (Sections 5 and 6).

2 Partial Latin Square Extension as Graph Coloring

This section illustrates how the partial Latin square extension problem can be considered as a graph coloring problem. This approach was first used in [14] with a great success to solve the related Latin square completion problem. However, two specific and significant features of the partial Latin square extension problem were ignored until now. We discuss them at the end of this section, which also provide additional motivations for this work.

2.1 Partial Latin Square Extension to Latin Square Graph

Given a Latin square $\mathcal{L}$ of order n composed of $n \times n$ cells, it can be transformed into a graph $G = (V, E)$, called a Latin square graph, with the set of vertices $V = n, \ldots, n$ cells of size $|V| = n^2$ and the set of edges E of size $|E| = n^2(n-1)$ where $\{u, v\} \in E$ if and only if u and v are two vertices representing two cells of the same row or the same column of $\mathcal{L}$ [13,14]. We can then solve the PLSE by finding a legal partial n-coloring (also called list coloring [13]) of the graph G using the colors in $\{1, ..., n\}$ while maximizing the number of colored vertices (or equivalently minimizing the number of uncolored vertices).

Let $D(v)$ denote the color domain of vertex v (i.e., the set of colors that can be used to color v). If v corresponds to a cell pre-filled with symbol k ($k \in \{1, ..., n\}$), $D(v) = \{k\}$. If v corresponds to an empty cell, v can receive a color in $\{1, ..., n\}$ or remain uncolored, indicated with the color 0. In other words, $D(v) = \{0, 1, ..., n\}$ for any vertex v representing an empty cell. Then a (partial) legal n-coloring of the associated Latin square graph G is a function $S : V \rightarrow \{D(v_1), \ldots, D(v_{|V|})\}$ such that for any pair of vertices u and v, if $S(u) \neq 0$, $S(v) \neq 0$, and they are linked by an edge ($\{u, v\} \in E$), then their colors $S(u)$ and $S(v)$ must be different ($S(u) \neq S(v)$). Note that a vertex receiving color 0 indicates an uncolored vertex.

A legal solution of the PLSE can also be seen as a partition of V into n independent sets V_1, V_2, ..., V_n and a set $V_0 = V \setminus \cup_{i=1}^{n} V_i$, such that V_i is the set of vertices receiving color i. A set V_i ($i = 1, \ldots, n$) is an independent set if $\forall (u, v) \in V_i, \{u, v\} \notin E$. An independent set is also called a color class.

Let $S = \{V_0, V_1, V_2, ..., V_n\}$ be a partition of the vertex set V, the objective of the partial Latin square extension problem (PLSE) in terms of the list-coloring problem can be stated as follows:

$$\text{minimize } f(S) = |V_0|, \tag{1}$$
$$\text{subject to } \forall u, v \in V_i, \{u, v\} \notin E, i = 1, 2 \ldots, n, \tag{2}$$

where the objective (1) is to minimize the cardinality of the set V_0 (number of uncolored vertices) and the constraints (2) ensure that the partition $\{V_0, V_1, V_2, \ldots, V_n\}$ is a legal but potentially partial n-coloring. Notice that this formulation of the partial Latin square extension problem can also be used to solve the Latin square completion problem (LSC), for which a legal solution S with $f(S) = 0$ is sought.

The constraints (2) can be reformulated with a constraint function c which simply counts the number of conflicts in S:

$$c(S) = \sum_{\{u,v\} \in E} \delta_{uv}, \tag{3}$$

where

$$\delta_{uv} = \begin{cases} 1 & \text{if } u \in V_i,\ v \in V_j,\ i = j \text{ and } i \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

If $\delta_{uv} = 1$, u and v are two conflicting vertices (i.e., they receive the same colors while they are adjacent in the graph). Clearly, a coloring S with $c(S) = 0$ corresponds to a legal n-coloring.

Figure 2 shows a PLSE instance (left), its Latin square graph (middle) and a legal partial coloring of the Latin square graph with two uncolored vertices (right).

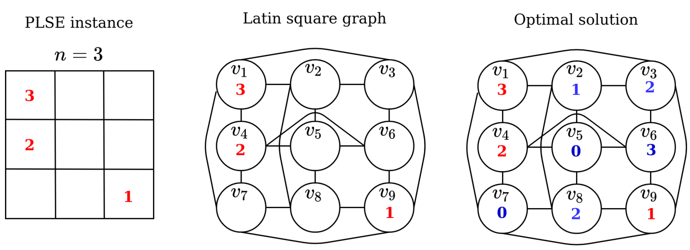

Fig. 2. Example of converting a partial Latin square extension instance (left) to a Latin square graph (middle) and an optimal partial coloring with two uncolored vertices (color 0) (right).

2.2 Preprocessing of the Latin Square Graph

As mentioned in [14], a preprocessing procedure can be applied to reduce a Latin square graph by removing the colored vertices (i.e., the filled cells). Indeed, if a vertex v of the graph represents a cell pre-filled with the symbol k in $1, \ldots, n$, the vertex definitely receives this single color k and can be removed from the graph. Moreover, since the color k cannot be assigned to any vertex

u adjacent to v (i.e. $\{u, v\} \in E$), this color can therefore be removed from the color domain $D(u)$.

Nevertheless, during the preprocessing, if the color domain of a vertex u becomes the singleton $D(u) = \{0\}$, it means that the corresponding cell cannot be filled. This cell remains definitively unfilled and the vertex u is removed from the graph. If one denotes by l the number of cells impossible to fill after this preprocessing phase, $n^2 - l$ defines an upper bound of the optimal value (score) of the given PLSE instance. For the special case of $l = 1$, a better upper bound is in fact $n^2 - 2$, as there is no optimal solution for a PLSE instance with a score of $n^2 - 1$ (cf. Theorem 6 in [19]).

The preprocessing procedure is described in ALgorithm 1. Its algorithmic complexity is in $O(|V|^2)$, where $|V|$ is the number of vertices in the original Latin square graph.

Algorithm 1 Preprocessing procedure for graph reduction of the PLSE problem

Input: A Latin square graph $G = (V, E)$ with some vertices already colored, each vertex v's color domain $D(v)$.
Output: A reduced graph and the number l of cells impossible to fill.

for each vertex $v \in V$ with singleton color domain $D(v) = \{k\}$ **do**
  $V \leftarrow V - \{v\}$ // Remove this colored vertex v from the graph
  $E \leftarrow E - \{\{u, v\} \in E\}$ // Remove the edges linked to v.
  for each uncolored $u \in V$ adjacent to v **do**
    $D(u) \leftarrow D(u) - \{k\}$ // Remove color k from the color domain $D(u)$
  end for
end for

$l = 0$
for each $v \in V$ **do**
  if $D(v) = \{0\}$ **then**
    $l = l + 1$
    $V \leftarrow V - \{v\}$ // Remove this node impossible to color
    $E \leftarrow E - \{\{u, v\} \in E\}$ // Remove the edges linked to v.
  end if
end for

Figure 3 (Right) displays the reduced graph of the Latin square graph shown in Figure 2. Numbers in accolades indicate the color domain $D(v_i)$ of each vertex v_i. In addition to the three precolored vertices v_1, v_4, v_9, vertex v_7 is also removed because its color domain is $D(v_7) = \{0\}$. Therefore, $l = 1$, leading to an upper bound $3^2 - 2 = 7$. Since this upper bound is equal to the lower bound of the two solutions in Figure 1, these two solutions are proven to be optimal for the given PLSE instance (i.e., a maximum of 7 filled cells /

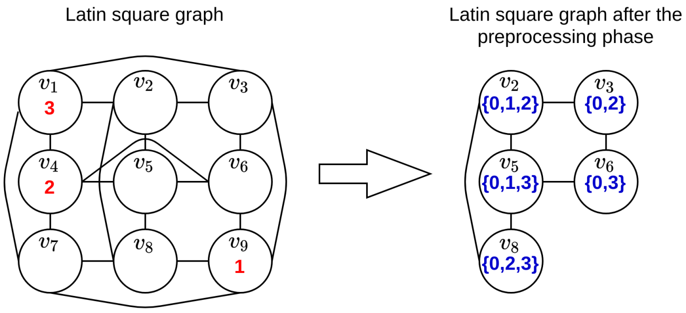

Fig. 3. Preprocessing of a Latin square graph with $n = 3$.

colored vertices or a minimum of 2 unfilled cells / uncolored vertices).

2.3 Special Features of the Transformed Coloring Problem

One observes two special features of the graph coloring problem transformed from the PLSE.

First, the Latin square graph coloring problem is a list-coloring problem [13], where the permissible colors of a vertex are limited to a list of colors in $\{0, 1, \ldots, n\}$, instead of the whole set $\{0, 1, ..., n\}$. Therefore, contrary to the standard graph coloring problem, candidate solutions are in general not invariant by permutation of colors. For example, in the legal partial coloring shown in Figure 2 on the right, it is impossible to swap colors 2 and 3 as the color 2 is not in the domain of the vertex v_6. Moreover, even a permissible color exchange between two colorings is not generally neutral. For example, consider the two legal solutions S_1 and S_2 displayed in Figure 4, where the pre-filled colors are in red, assigned colors are in blue and possible color changes are in green. The solution S_2 is the same as the solution S_1 except that the colors 1 and 3 are swapped. After this swap, it becomes impossible to change the color of the vertex v_2 in S_2 while it was possible in S_1. S_1 and S_2 are thus two different candidate solutions for the PLSE, while they represent the same coloring for the conventional graph coloring problem. This observation implies that for this list-coloring problem, solutions are not invariant by permutation of the colors. As a result, the so-called set-theoretic partition distance [20], which is usually used to measure the distance between two solutions for graph coloring [21,22], is not meaningful for the list-coloring problem. Instead, the Hamming distance D^H is more suitable to measure the distance between solutions for our coloring problem (cf. Section 3.4).

Secondly, the partial list-coloring from the PLSE aims to find a legal coloring such that the objective function $f(S)$ defined by equation (1) (number of uncolored vertices) is minimized. Therefore, it is critical that the algorithm is able to decide which vertices are to be left uncolored when it is impossible to

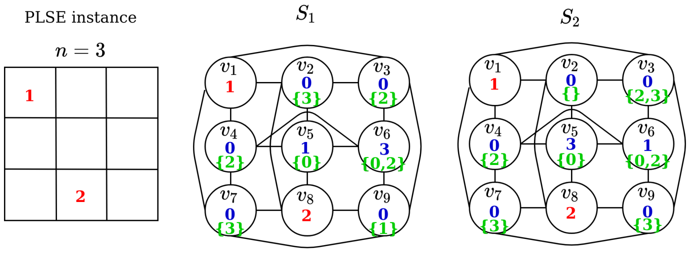

Fig. 4. Two legal solutions S_1 and S_2 of the PLSE instance. The two solutions are the same except that the colors 1 and 3 are swapped.

color all the vertices of the graph.

For these reasons, we introduce an algorithm specifically designed to solve the partial list-coloring problem of Latin square graphs of the PLSE. This algorithm, presented in the next section, can also be applied to solve the related Latin square completion problem (LSC).

3 Massively Parallel Memetic Algorithm

We describe in this section the massively parallel memetic algorithm (MPMA) for coloring Latin square graphs.

3.1 Search Space and Evaluation Function

The enlarged search space Ω explored by the MPMA algorithm is composed of the legal, illegal and potentially partial candidate solutions.

Let $G = (V, E)$ be the reduced Latin square graph with $|V|$ vertices $\{v_1, \ldots, v_{|V|}\}$, and color domains $D(v_i) \subseteq \{0, 1, ..., n\}$ ($i = 1, \ldots, |V|$) obtained after the pre-processing phase. Then the space Ω is given by

$$\Omega = \{S : V \to \{D(v_1), \ldots, D(v_{|V|})\}\}. \tag{5}$$

The MPMA algorithm aims to find a legal, possibly partial solution S (with $c(S) = 0$) of the Latin square graph with the minimum number of uncolored vertices $f(S)$ (for functions f and c, see Section 2.1).

We define the following extended evaluation function F (to be minimized) to assess the quality (fitness) of a candidate solution $S \in \Omega$:

$$F(S) = f(S) + \phi \times c(S), \quad (6)$$

where $\phi > 0$ is a penalty parameter controlling the impact of the constraint function c on the overall score. Generally, decreasing the value of ϕ favors solutions with less uncolored vertices and more conflicts, while increasing its value promotes legal (conflict-free) and partial colorings. If ϕ is set to the value of 1, x uncolored vertices and x conflicts contribute equally to the quality of the solution.

3.2 Main Scheme

The proposed MPMA algorithm is based on the population-based memetic framework [23], which has been applied to graph coloring problems [18,22,24]. It should be noted that these memetic algorithms typically use a small population of no more than 20 individuals and are elitist evolutionary algorithms. As such, each generation typically creates one or two offspring solutions via a crossover operator, which are then improved by a local search procedure.

The massively parallel memetic algorithm proposed in this work uses a very large population P ($|P| \geq 10^4$), whose individuals evolve in parallel in the search space. This approach ensures a high degree of diversity in the population, which favors a large exploration of candidate solutions. In order to take advantage of this large population, we use the computational power of modern GPUs to perform parallel computations at each generation: local searches, distance evaluations and crossovers. The only part that remains sequential is the population update operation that merges the current population and the offspring population to create the next population.

The algorithm takes as input a reduced Latin square graph G (see Section 2.2) and tries to find a legal, possibly partial, coloring with a minimum number of uncolored vertices. The pseudo-code of MPMA is presented in the algorithm 2, while its flowchart is displayed in Figure 6. At the beginning, all the individuals of the population are randomly initialized in parallel, which are improved by local search at the beginning of the first generation of the algorithm (see below and Figure 6). Then, the algorithm repeats a loop (generation) until a stopping criterion (for example, a time limit or a maximum number of generations) is satisfied. Each generation t involves the execution of four components:

(1) The p individuals (illegal n-colorings) of the current population are simultaneously enhanced by running a two-phase local search in parallel (see Section 3.3) to minimize the fitness function f (uncolored vertices) and the constraint function c (conflicting vertices).

Algorithm 2 Massively parallel memetic algorithm for Latin square graph coloring

Input: Reduced Latin square graph $G = (V, E)$, population size p, color domain $D(v)$ of each vertex $v \in V$.
Output: The best legal partial coloring S^* found
$P = \{S_1, \dots, S_p\} \leftarrow$ population_initialization
$S^* = \emptyset$ and $e(S^*) = |V|$.
$\{S_1^O, \dots, S_p^O\} \leftarrow \{S_1, \dots, S_p\}$
repeat
  for $i = \{1, \dots, p\}$, **in parallel do**
    $S'_i \leftarrow$ two-phase_local_search(S_i^O) /∗ Section 3.3
  end for
  $S'^* = \text{argmin}\{f(S'_i), i = 1, \dots, p\}$
  if $f(S'^*) < f(S^*)$ **then**
    $S^* \leftarrow S'^*$
  end if
  $D \leftarrow$ distance_computation$(S_1, \dots, S_p, S'_1, \dots, S'_p)$ /∗ Section 3.4
  $\{S_1, \dots, S_p\} \leftarrow$ pop_update$(S_1, \dots, S_p, S'_1, \dots, S'_p, D)$ /∗ Section 3.4
  $\{S_1^O, \dots, S_p^O\} \leftarrow$ build_offspring$(S_1, \dots, S_p, D)$ /∗ Section 3.5
until stopping condition met
return S^*

(2) The distances between all pairs of existing individuals and the individuals improved by local search are computed in parallel (see Section 3.4).
(3) Then, the population update procedure (see Section 3.4) merges the $2p$ existing and new individuals to update the population, taking into account the fitness f of each individual (number of uncolored vertices) and the distances between individuals in order to maintain a healthy diversity in the population.
(4) Finally, each individual is matched with its nearest neighbor in the population and p crossovers are run in parallel to generate p offspring solutions (see Section 3.5), which are improved by parallel iterated local search during the next generation $(t + 1)$.

The algorithm stops when a predefined time condition is reached or an optimal solution S^* is found. S^* is an optimal solution if 1) $c(S^*) = 0$, $f(S^*) = 0$, and $l \neq 1$ (i.e., all empty cells are filled), or 2) $c(S^*) = 0$, $f(S^*) = 1$, and $l = 1$ (the tightest upper bound is reached, see Section 2.2). If the algorithm does not find an optimal solution when it stops, it returns the best legal solution S^* (with $c(S^*) = 0$) found so far, with a number of unfilled cells $f(S^*) > 0$. Then the score $n^2 - l - f(S^*)$ is a lower bound of the given PLSE instance.

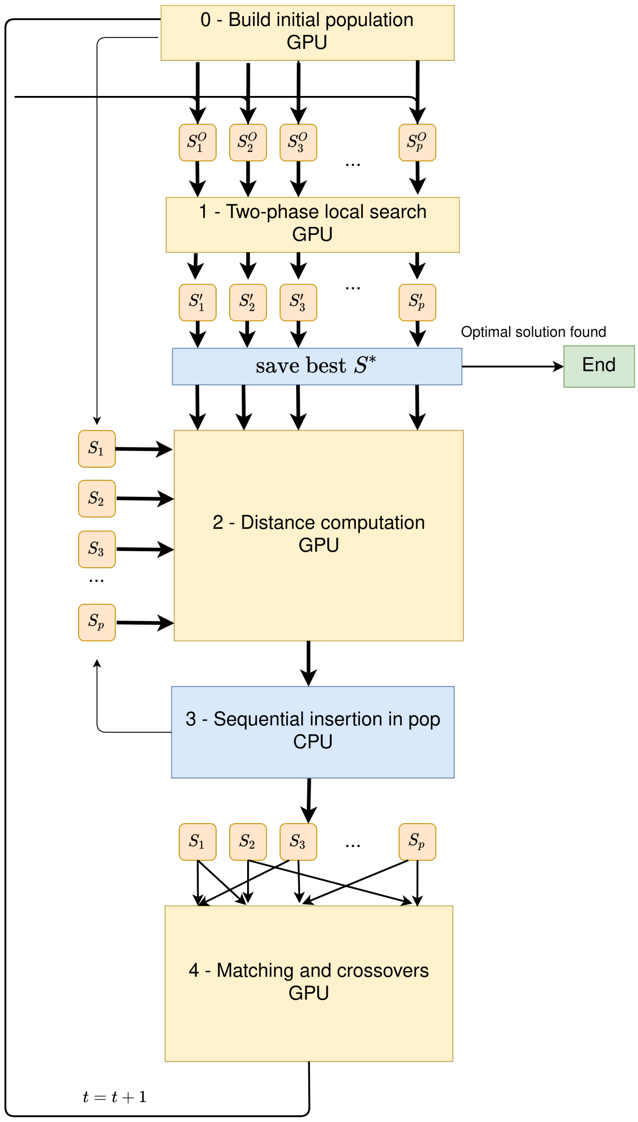

Fig. 5. General scheme of the MPMA algorithm.

3.3 Parallel Two-phase Local Search

MPMA employs a two-phased partial legal and illegal tabu search (PLITS) to simultaneously improve in parallel the individuals of the current population. Specifically, PLITS relies on the tabu search metaheuristic to explore candidate solutions of the space Ω guided by the extended fitness function F given by equation (6). Indeed, tabu search is a popular method for graph coloring [25–27] and often used as the local optimization components of memetic algorithms [14,22,28].

Given a solution $S = \{V_0, V_1, V_2, ..., V_n\}$, PLITS uses the one-move operator to displace a vertex v from its current color class V_i to a different color class V_j such that $i \neq j$ and $j \in D(v)$, leading to a neighboring solution denoted as $S\oplus < v, V_i, V_j >$. Let $\mathcal{C}(S)$ be the set of conflicting vertices in S, i.e., $\mathcal{C}(S) = \{v \in V_i : 1 \le i \le n, \exists u \in V_i, (u,v) \in E, u \neq v\}$. To make the examination of candidate solutions more focused, PLITS only considers the uncolored vertices in V_0 and conflicting vertices in $\mathcal{C}(S)$ for color changes.

The one-move neighborhood applied to the uncolored vertices of S is given by

$$N_0(S) = \{S\oplus < v, V_0, V_j >: v \in V_0, 1 \le j \le n, j \in D(v)\}.$$

The one-move neighborhood applied to the conflicting vertices of S is given by

$$\begin{aligned} N_c(S) = \{&S\oplus < v, V_i, V_j >: v \in \mathcal{C}(S), v \in V_i, 1 \le i \le n, \\ &0 \le j \le n, j \in D(v), i \neq j\}. \end{aligned}$$

Notice that a conflicting (colored) vertex can be moved to the set V_0 by the one-move operator, becoming thus uncolored.

PLITS explores the global one-move neighborhood:

$$N(S) = N_0(S) \cup N_c(S). \tag{7}$$

PLITS makes transitions between the various n-partial colors with the help of the neighborhood $N(S)$ and the extended evaluation function F. At each iteration PLITS replaces the current solution S with the best eligible neighboring solution S' taken from $N(S)$. After each iteration, the corresponding one-move is stored in the tabu list to prevent the search from returning to a previously visited solution for the next T iterations (tabu tenure). Following [28], the tabu tenure depends on the number of vertices eligible for the one-move operator (i.e., $|V_0| + |\mathcal{C}(S)|$ in our case) and is set to the value of $L + \alpha(|V_0| + |\mathcal{C}(S)|)$), where L is a random integer from $[0; 9]$ and α is a parameter fixed at 0.6. A neighboring solution S' is considered admissible if it is not prohibited by the tabu list or if it is better (according to the extended function F) than the best solution found so far. Neighborhood evaluations are performed incrementally like in [28]. As the algorithm 3 shows, we run the PLITS procedure in parallel on the GPU to increase the quality of the current population. The time complexity of this PLITS procedure is in $O(|V| \times n \times nbIter_{TS} \times p)$. The space complexity of the PLITS procedure is in $O(|V| \times n \times p)$ (size of the tabu tenure matrices for all the individuals of the population).

The PLITS procedure is performed in two phases with different search focuses.

Algorithm 3 Parallel partial legal and illegal tabu search

```
Input: Population $P = \{S_1, \ldots, S_p\}$, depth of tabu search $nbIter_{TS}$, color domain $D(v)$ of each vertex $v \in V$.
Output: Improved population $P' = \{S_1'^*, \ldots, S_p'^*\}$.
for $i = \{1, \ldots, p\}$, in parallel do
    $S_i'^* \leftarrow S_i$    /* Records the best solution found so far on each local thread.
end for
$iter = 0$
for $i = \{1, \ldots, p\}$, in parallel do
    for $t = \{1, \ldots, nbIter_{TS}\}$ do
        Choose a neighboring solution $S_i' \in N(S_i)$ which is not forbidden by the tabu list or better than $S_i$ (according to the extended evaluation function $F$).
        $S_i \leftarrow S_i'$
        if $F(S_i') < F(S_i'^*)$ then
            $S_i'^* \leftarrow S_i'$
        end if
    end for
end for
return $P' = \{S_1'^*, \ldots, S_p'^*\}$
```

The first phase favors a large exploration of candidate solutions by setting ϕ to the value of 0.5 and performs $nbIter_{TS} = 100 * |V|$ iterations. The second phase focuses on resolving the conflicts in the solutions of the population to obtain P legal colorings (with $c(S) = 0$). For this purpose, ϕ is set to the large value of $|V|$ during $nbIter_{TS} = 2 * |V|$ iterations.

After the local search, the best coloring $S_i'^*$ among the p conflict-free colorings in terms of the objective function f is used to update the recorded best solution S^* if $S_i'^* < S^*$.

3.4 Population Update

The p new legal colorings from the PLITS procedure are used to update the population. For this, MPMA maintains a $p \times p$ matrix to record all the distances between any two solutions of the population. This symmetric matrix is initialized with the $p \times (p-1)/2$ pairwise distances computed for each pair of individuals in the initial population, and then updated each time a new individual is inserted in the population.

To merge the p new solutions and the p existing solutions, MPMA needs to evaluate (i) $p \times p$ distances between each individual in the population $P = \{S_1, \ldots, S_p\}$ and each improved offspring individual in $P' = \{S_1', \ldots, S_p'\}$ and (ii) $p \times (p-1)/2$ distances between all the pairs of individuals in P'. All the

Algorithm 4 Sequential population update procedure

Input: Population $P_t = \{S_1, \dots, S_p\}$ (generation t) and offspring population $P' = \{S'_1, \dots, S'_p\}$ (generation t)
Output: Updated population P_{t+1} (generation $t+1$)
$P_{t+1} = \emptyset$ /∗ Initilize new population
$P^{all} = P_t \cup P'$ /∗ Merge existing and improved new solutions
$S^{best} = \text{argmin}_{S \in P^{all}}\ e(S)$ /∗ Identify the best legal solution in P^{all}
$P_{t+1} = P_{t+1} \cup \{S^{best}\}$ /∗ Add S^{best} in P_{t+1}
$P^{all} = P^{all} \setminus \{S^{best}\}$ /∗ Remove S^{best} from P^{all}
/∗ Add n-colorings in P_{t+1} until it contains the p best solutions of P^{all} with the condition that $D^H(S_i, S_j) > |V|/10$, for all $S_i, S_j \in P_{t+1}$, $i \neq j$
while $|P_{t+1}| < p$ **do**
  $S^{best} = \text{argmin}_{S \in P^{all}}\ e(S)$
  $dist = \min_{A \in P_{t+1}}\ D(S^{best}, A)$
  if $dist > |V|/10$ **then**
    $P_{t+1} = P_{t+1} \cup \{S^{best}\}$
    $P^{all} = P^{all} \setminus \{S^{best}\}$
  end if
end while
return P_{t+1}

$p \times p + p \times (p-1)/2$ distance computations are independent from one another, and are performed in parallel on the GPU (one computation per thread).

Given two colorings S_i and S_j, MPMA uses the Hamming distance $D^H(S_i, S_j)$ to measure the dissimilarity between S_i and S_j, which corresponds to the number of vertices that are colored differently in S_i and S_j:

$$D^H(S_i, S_j) = |\{v \in V, S_i(v) \neq S_j(v)\}|. \tag{8}$$

The complexity of the distance computations for the whole population is in $O(|V| \times p^2)$.

Following [21], the population update procedure of MPMA aims to keep the best individuals, but also to ensure minimal spacing between individuals. The population update procedure (Algorithm 4) greedily adds one by one the best individuals of $P^{all} = \{S_1, \dots, S_p\} \cup \{S'_1, \dots, S'_p\}$ into the population of the next generation P_{t+1} until P_{t+1} reaches p individuals, so that $D^H(S_i, S_j) > |V|/\gamma$ ($\gamma > 1,0$ is a parameter), for any $S_i, S_j \in P_{t+1}$, $i \neq j$. The time complexity of the population update procedure is in $O(p^2)$. In practice for an instance of medium size (reduced Latin square graph with about $|V| = 750$ vertices), this population update procedure is executed in a time corresponding to roughly 3% of the time spent in the local search procedure at each generation. The space complexity of this procedure is in $O(|V|p + p^2)$ (due to the distance matrices storage).

3.5 Parent Matching and Crossover

At each generation, the MPMA algorithm performs in parallel p crossovers to generate p offspring solutions. For this, MPMA uses each existing solution in the current population as the first parent and selects another existing solution as the second parent with a specific parent matching strategy. The idea is to ensure that each individual in the population has a chance to transmit some *genetic information* to the next generation while encouraging the creation of diversified offspring.

3.5.1 Parent Matching Strategy

The population update strategy presented in the last section ensures that the individuals in the next population are high quality, but also sufficiently distant. This property provides a first basis for ensuring that for each of the p crossovers, we can find a second parent that is sufficiently distant from the first parent. This helps to build diverse offspring solutions that are different from their parents, and thus helps the algorithm to continuously explore new areas in the search space.

However, as we use a very large population, individuals can be highly different and share very little information. Indeed, we experimentally observed that the average pairwise distance in the population is usually very large, around $0.7 \times |V|$ even after many generations. Meanwhile, a study in [22] showed that for the standard graph coloring problem, crossing-over two highly different parents results in offspring of poor quality because no meaningful shared information can be transmitted from parents to offspring.

Thus, for each individual S_i (i.e., the first parent), we choose, among the other individuals in the population, the nearest neighbor S_j in the sense of the precomputed Hamming distance D, as the second parent. The time complexity of the matching procedure is in $O(p^2)$.

3.5.2 Parameterized Asymmetric Uniform Crossover

The popular greedy partition crossover (GPX) [28] and its variants have proven to be very successful for the graph coloring problem [18,22,29]. GPX was also adapted to the related LSC, leading to the maximum approximate group based crossover (MAGX) [14]. However, the GPX crossover has some limitations for the PLSE due to the fact that solutions are not invariant by permutations of color groups (cf. Section 2.3) and high-quality solutions do not share significant backbones (they are far away from each other, see Section 5).

For the PLSE, we introduce a parameterized asymmetric uniform crossover (AUX), which is easy to compute for a very large population of individuals and allows the transmission of favorable parental features to the next generation.

Given a first parent S_i and a second parent S_j, an offspring solution S_i^O is built such that each vertex v receives the color of S_i with probability p_{ij} and the color of S_j with probability $1-p_{ij}$. The probability p_{ij} depends proportionally on the Hamming distance between the parents S_i and S_j and is given by

$$p_{ij} = 1 - \frac{|V|}{\beta \times D^H(S_i, S_j)}, \tag{9}$$

where $\beta > 1.0$ is a real parameter controlling the degree of diversity of the resulting offspring. The complexity of computing AUX crossovers for the entire population is in $O(|V| \times p)$.

As $|V|/\gamma$ is the minimum spacing between two individuals in the population (cf. Section 3.4), we set β such that $\beta > \gamma > 0$, in order to have $\forall i, j \in [\![1, \ldots, p]\!]^2, i \neq j,\ |V|/\beta < D^H(S_i, S_j)$. This ensures that $\forall i, j \in [\![1, \ldots, p]\!]^2,\ 0 < p_{ij} < 1$.

Notice that when p_{ij} is fixed to the value of 0.5, we obtain the classical Uniform Crossover (UX) [30]. With the UX crossover, the resulting offspring is on average equidistant from both parents. However, as we empirically show in Section 4, the UX crossover does not work well for the PLSE (it is too much disruptive). The proposed AUX crossover uses the probability p_{ij} to make itself more conservative by considering the distance between two parents. Specifically, if two parents are similar (with a small distance), the offspring can equally inherit information from the parents. On the contrary, if the parents are very different (with a large distance), it is preferable to conserve more information from one parent (the first parent) to avoid an offspring solution that is far away from both parents. AUX achieves this goal by adjusting the coefficient β which influences the probability.

For two given parents S_i and S_j, the expected distance between the offspring S_i^O and its first parent S_i is $\bar{D}^H(S_i, S_i^O) = |V|/\beta$. The expected distance between the offspring S_i^O and its second parent S_j is $\bar{D}^H(S_j, S_i^O) = D^H(S_i, S_j) - |V|/\beta$. If we choose $\beta \geq 2\gamma$, $\bar{D}^H(S_i, S_i^O) \geq \bar{D}^H(S_j, S_i^O)$ always holds. As such, in average the child preserves more genetic information from the first parent compared to the second parent. Given that MPMA uses every individual in the current population as the first parent, all individuals are offered the same chance to transmit a large part of their genetic information to their offspring, leading to a large coverage of the search space.

Figure 6 illustrates the creation of six offspring solutions $\{S_i^O\}_{i=1}^6$ (in red) generated from the population $\{S_i\}_{i=1}^6$ (in black). In this case, the offspring

Algorithm 5 Parallel asymmetric uniform crossover AUX

Input: Population $P = \{S_1, \ldots, S_p\}$, with $S_i = (V_0^i, V_1^i, \ldots, V_n^i)$, for $i = 1, \ldots, p$.
Output: Offspring population $P^O = \{S_1^O, \ldots, S_p^O\}$
for $i = 1, \ldots, p$, in parallel **do**
  $S_j \leftarrow$ Find and make a copy of the nearest neighbor of S_i from P according to the distance D such that $i \neq j$ and such that this crossover (i, j) has not been tested yet.
  $p_{ij} = 1 - \frac{|V|}{\beta \times D^H(S_i, S_j)}$
  for $l = \{1, \ldots, |V|\}$ **do**
    With probability p_{ij}, $S_i^O(v_l) = S_i(v_l)$
    Otherwise $S_i^O(v_l) = S_j(v_l)$
  end for
end for
return P^O

S_1^O to S_6^O are respectively generated from the ordered pairs of parents (S_1, S_2), (S_2, S_3), (S_3, S_4), (S_4, S_5), (S_5, S_4), (S_6, S_1).

As one notices, each offspring is situated in between its two parents in the search space and always closer to its first parent (in terms of the Hamming distance). The norm of each translation vector is equal to $|V|/\beta$ in average.

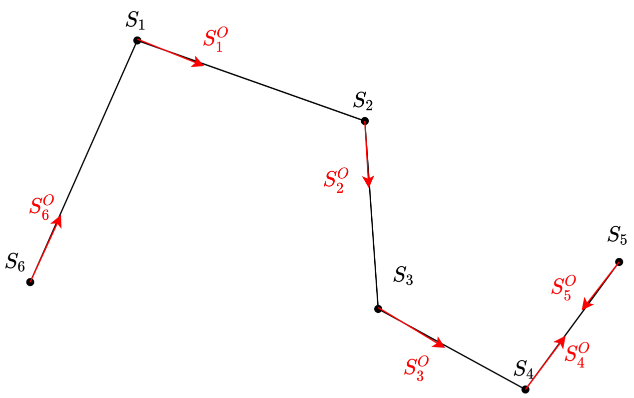

Fig. 6. Resulting offspring individuals $\{S_i^O\}_{i=1}^6$ (in red) generated from the population $\{S_i\}_{i=1}^6$ (in black).

The overall parent matching and the AUX crossover are summarized in Algorithm 5. All the p crossover operations are performed in parallel on individual GPU threads. The time and space complexities of the crossover procedure are in $O(|V|p)$.

3.6 Implementation on Graphic Processing Units

MPMA was programmed in Python with the Numba library for CUDA kernel implementation. It is specifically designed to run on GPUs. In this work we used a V100 Nvidia graphic card with 32 GB memory. The source code of the algorithm is available at `https://github.com/GoudetOlivier/MPMA_code`.

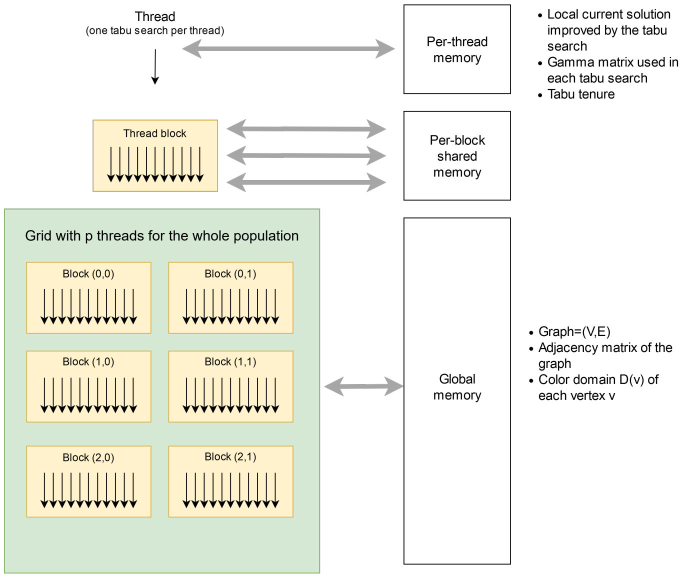

Fig. 7. Parallel tabu searches launched on GPU grid.

Figure 7 shows the organization of threads on the GPU grid and the memory hierarchy on the GPUs used to execute the p tabu searches in parallel for the entire population each generation. Each of the p tabu searches (see Section 3.3) is executed on a single thread. For fast memory access, a local memory per thread is used to store specific local information such as the current solution and tabu tenure. Threads are grouped in blocks of size 64 and launched on the GPU grid. A global memory is used to store general graph information such as the graph adjacency matrix and the color domain of each vertex to avoid duplication of information. All these p tabu searches are run with a CUDA kernel function and the best results obtained in each tabu search are transferred to the CPU after synchronization.

The same type of kernel function on the GPUs is used to compute in parallel the p distance calculations (see Section 3.4) and the p crossovers (see Section 3.5) at each generation. However, some operations such as the best solution saving procedure and the population update procedure (cf. Section 3.4) are performed on the CPU because they cannot be parallelized.

3.7 A variant of the Algorithm for Highly Constrained Instances

As shown in Section 4, the MPMA algorithm excels on under-constrained to moderately over-constrained PLSE instances with a filled ratio r below 80%. However, its performance slightly deteriorates on highly constrained instances when $r \geq 80\%$. For these cases, we observed that better results can be reached by directly minimizing the number of uncolored vertices (i.e., fitness f of Section 2.1) in the space of legal (i.e., conflict-free) partial colorings. For these highly constrained instances, we create a simplified MPMA variant called Partial-MPMA that works with legal partial colorings (instead of conflicting colorings) and makes the following two changes in MPMA.

- A greedy conflict removal procedure is applied to repair each offspring solution into a legal partial coloring. For this, the vertex which is conflicting with the largest number of vertices is uncolored first (i.e., reassigned the color 0), followed by the vertex with the second largest conflicts and so on. This process continues until a partial conflict-free coloring is reached.
- The two-phase tabu search procedure of Section 3.3 is replaced by the PartialCol coloring algorithm of [31] adapted to the list-coloring problem. This PartialCol algorithm uses tabu search to explore the space of legal partial colorings by minimizing the number of uncolored vertices.

4 Experimental Results

This section is dedicated to a computational assessment of the MPMA algorithm for solving the partial Latin square extension problem, by making comparisons with the state-of-the-art methods. Additional results are presented in Appendix B for the related Latin square completion problem.

4.1 Benchmark Instances

We carried out extensive experiments on the 1800 PLSE benchmark instances introduced in [12]. These instances are parametrized by the grid order $n \in \{50, 60, 70\}$ and the ratio $r \in \{0.3, 0.4, \ldots, 0.8\}$ of pre-filled cells in the $n \times n$ grid. Given (n, r) and starting from an empty $n \times n$ grid, a PLSE instance was constructed by repeatedly assigning a different symbol in an empty cell chosen randomly so that the Latin square condition is respected and until $r \times n^2$ cells are assigned symbols. For each (n, r) combination, 100 instances are available. Note that such a PLSE instance does not always admit a complete solution (i.e., some cells must be left unfilled). This is typically the case for relatively strongly constrained instances when $r > 60$ (i.e., when at least 60% cells are pre-filled). Moreover, as shown in [9,12], under-constrained instances ($r \leq 0.5$) and over-constrained instances ($r > 0.7$) are easier than medium-constrained instances with r between 0.6 and 0.7.

It is clear that n^2 is an upper bound for these instances (all cells are filled). When the grid cannot be fully filled, a safe upper bound is given in [19], corresponding to $n^2 - 2$ (all but 2 cells are filled). This bound indicates that if a grid cannot be completed, at least two cells will be left unfilled.

Like [14], we first convert these instances to Latin square graphs and apply the preprocessing algorithm of Section 2.2 to reduce them, leading to graphs with less than 500 vertices for $(n, r) = (50, 0.8)$ and up to 3430 vertices for $(n, r) = (70, 0.3)$. The preprocessing takes no more than several seconds.

4.2 Parameter Setting

The population size p of MPMA is set to $p = 12288$, which is chosen as a multiple of the number of 64 threads per block. This large population size offers a good performance ratio on the Nvidia V100 graphics card that we used in our experiments, while remaining reasonable for pairwise distance calculations in the population, as well as the memory occupation on the GPU, especially when solving very large instances. Indeed the overall space complexity of the proposed algorithm is in $O(|V| \times n \times p + p^2)$. It is in particular quadratic with respect to the size p of the population. A sensitivity experiment of the results with respect to the population size is presented in Section 5. In addition to the population size, the parameter α of the tabu search is set to its classical value of 0.6 and the number of tabu iterations $nbIter_{TS}$ depends on the size $|V|$ of the graph. The parameter γ for the minimum spacing between two individuals is set to 10. The parameter β for adjusting the distance of the offspring from their parents is fixed at 20.

Table 1 summarizes the parameter setting, which can be considered as the default and is used for all our experiments.

Table 1
Parameter setting in MPMA

Parameter	Description	Value
p	Population size	12288
$nbIter_{TS}$	Number of iterations tabu search	$100 \times \|V\|$
α	Tabu tenure parameter	0.6
γ	Parameter for the spacing between two individuals	10
β	Parameter for the generation of offspring	20

4.3 Comparative Results on the Set of 1800 PLSE Instances

This section shows a comparative analysis on the 1800 PLSE instances with respect to the state-of-the-art methods. Given the stochastic nature of the MPMA algorithm, each instance is independently solved 5 times.

Table 2 summarizes the computational results of MPMA compared to the best results in the literature reported in [12,14]. For each instance MPMA was launched with a maximum of 100 billions of tabu search iterations. The reference methods include the 7 PLSE approaches in [12]: CPX-IP, CPX-CP, LSSOL, 1-ILS*, 2-ILS, 3-ILS and Tr-ILS*, where CPX-IP and CPX-CP are exact Integer Programming and Constraint Programming solvers from IBM/ILOG CPLEX, LSSOL denotes the tool LocalSolver. 1-ILS*, 2-ILS, 3-ILS and Tr-ILS* are four iterated local search algorithms with three different neighborhoods. We cite the results of the recent MMCOL algorithm [14], which is designed for the related LSC problem and reported results on the 1800 PLSE instances with an adapted version of MMCOL. We also ran the FastLSC algorithm [15] with the default parameters provided by the authors. As FastLSC is designed exclusively for the LSC problem, it does not provide any legal solution or even crashes for PLSE instances for which it is impossible to fill the grid completely. This happens for highly constrained instances, in general when $r \geq 0.7$.

Columns 1 and 2 of Table 2 show the characteristics of each instance (i.e., grid order $n \in \{50, 60, 70\}$ and ratio $r \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ of pre-assigned symbols). Columns 3-10 present the average number of filled cells in the best solutions obtained by the reference algorithms for the 100 instances of each type (n, r). The number in brackets indicates the number of instances for which the grid is completely filled. The results of the proposed MPMA algorithm and Partial-MPMA variant are reported in columns 11 and 12 respectively [1]. Bold numbers show the dominating values while a star indicates an optimal value (corresponding to the n^2 upper bound).

We observe that MPMA (standard version) always obtains the best scores (in bold) except for the over-constrained instances with $r = 0.8$. For the instances with $r = 0.8$, our Partial-MPMA variant always obtains the best results. For the loosely constrained or under-constrained instances with $r < 0.7$, the three compared algorithms (MPMA, MMCOL and FastLSC) can completely fill the grid for exactly the same number of instances. For the strongly constrained or over-constrained instances with $r \geq 70$, FastLSC fails to find a solution except for 4 instances with $n = 70$ and $r = 0.7$ for which it can fill the grid like MMCOL (against 5 instances for MPMA).

[1] The certificates of the best solutions of MPMA and Partial-MPMA for these 1800 instances are available at `https://github.com/GoudetOlivier/MPMA_code`

The best competitors, Tr-ILS*, MMCOL and FastLSC, were launched with a limited amount of available times in [12,14,15]: up to 10 seconds for Tr-ILS*, up to two hours for MMCOL and up to 1000 seconds for FastLSC. In order to verify if these algorithms can improve their results by using more computation time, we ran the codes of these three algorithms with a much relaxed time limit of 48 hours per run and per instance on Intel Xeon ES 2630, 2,66 GHz CPU. The results are shown in Table 3. For each compared algorithm, we report the best and average results over 5 runs (f_{best} and f_{avg}) as well as the average computation time needed to reach its best result.

With this much relaxed time limit, both Tr-ILS* and MMCOL indeed improve their-own results reported in [12] and [14] (also shown in Table 2). Meanwhile, they are still outperformed by MPMA/Partial-MPMA on the strongly constrained instances with $r \geq 0.7$. FastLSC also improves its performance and solves one more instance of set $n = 70$ and $r = 0.7$. Specifically, among the 100 instances with $n = 70$ and $r = 0.7$, FastLSC, like MPMA, completely fills the same set of 5 instances (with id 6, 14, 42, 44 and 99, see Table A.1) For the PLSE instances that can be completely filled, FastLSC is the fastest algorithm compared to MMCOL and MPMA.

For under-constrained (easy) instances, one notices that MPMA takes much more times to achieve its best results. This comes from the fact that every kernel operation launched on the GPU cannot be stopped until it is completed on each thread. Therefore, even if a solution of the instance is found in one thread, one still needs to wait for all the threads to finish their computation before retrieving the result. In fact, for these easy instances, a very large population with a high diversity is not really mandatory. MPMA can reach the optimal solutions faster with a much reduced population.

Table 2
Comparative results of MPMA and its Partial-MPMA variant with the state-of-the-art methods (CPX-IP, CPX-CP, LSSOL, 1-ILS*, 2-ILS, 3-ILS, Tr-ILS* in [12] and MMCOL in [14]) in terms of the average number of filled cells for each type of 100 PLSE instances of size $n \in \{50, 60, 70\}$ and ratio of pre-assigned symbols $r \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. Dominating results are indicated in bold. Note that no statistical tests are reported in this table because it is a comparison with the best bounds published in the literature. The percentage of instances for which the grid is completely filled is shown in parentheses.

Instance		CPX-IP	CPX-CP	LSSOL	1-ILS*	2-ILS	3-ILS	Tr-ILS*	MMCOL	FastLSC	MPMA	Partial-MPMA
n	r	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}
50	0.3	2496.03 (10)	2499.87 (98)	2496.35 (13)	**2500*** (100)	2499.98 (99)	2499.96 (98)	**2500*** (100)	**2500*** (100)	**2500*** (100)	**2500*** (100)	**2500*** (100)
	0.4	2493.78 (1)	2498.02 (66)	2494.65 (4)	2499.98 (99)	**2500*** (100)	2499.86 (93)	**2500*** (100)	**2500*** (100)	**2500*** (100)	**2500*** (100)	2493.35 (1)
	0.5	2488.52 (0)	2489.92 (4)	2492.96 (1)	2499.89 (95)	2499.95 (98)	2499.25 (67)	**2500*** (100)	**2500*** (100)	**2500*** (100)	**2500*** (100)	2491.82 (2)
	0.6	2476.21 (0)	2478.87 (0)	2489.21 (0)	2496.23 (7)	2496.3 (7)	2494.67 (0)	2497.18 (20)	2499.64 (85)	(85)	**2499.7** (85)	2485.64 (0)
	0.7	2446.4 (0)	2451.04 (0)	2463.45 (0)	2469.47 (0)	2469.78 (0)	2467.77 (0)	2470.07 (0)	2478.94 (0)	(0)	**2484.38** (0)	2466.95 (0)
	0.8	**2394.58** (0)	2388.1 (0)	2393.67 (0)	2394.14 (0)	2394.11 (0)	2394.09 (0)	2394.14 (0)	2364.61 (0)	(0)	2393.24 (0)	**2394.58** (0)
60	0.3	3593.07 (0)	3598.29 (77)	3593.2 (0)	3599.98 (99)	**3600*** (100)	3599.28 (64)	**3600*** (100)	**3600*** (100)	**3600*** (100)	**3600*** (100)	3597.56 (65)
	0.4	3590.68 (0)	3592.55 (19)	3591.17 (0)	3599.97 (99)	3599.96 (98)	3598.58 (43)	**3600*** (100)	**3600** (100)	**3600*** (100)	**3600*** (100)	3596.2 (23)
	0.5	3585.29 (0)	3585.83 (1)	3587.5 (0)	3599.65 (83)	3599.58 (81)	3597.53 (21)	3599.94 (97)	**3600*** (100)	**3600*** (100)	**3600*** (100)	3589.18 (2)
	0.6	3572.61 (0)	3573.7 (0)	3585.52 (0)	3595.82 (5)	3595.85 (2)	3592.77 (1)	3596.67 (13)	**3599.94** (97)	(97)	**3599.94** (97)	3578.9 (0)
	0.7	3534.71 (0)	3540.45 (0)	3561.05 (0)	3571.47 (0)	3570.58 (0)	3566.51 (0)	3572.12 (0)	3589.82 (0)	(0)	**3593.52** (0)	3556.65 (0)
	0.8	3478.58 (0)	3464.14 (0)	3476.44 (0)	3478.59 (0)	3478.37 (0)	3478.05 (0)	3478.49 (0)	3431.85 (0)	(0)	3477.80 (0)	**3480.03** (0)
70	0.3	4890.2 (0)	4893.75 (38)	4890.25 (0)	4899.98 (99)	4899.98 (99)	4897.32 (13)	**4900*** (100)	**4900*** (100)	**4900*** (100)	**4900*** (100)	4896.48 (55)
	0.4	4887.73 (0)	4888.36 (5)	4887.98 (1)	4899.96 (98)	4899.98 (99)	4896.4 (4)	4899.98 (99)	**4900*** (100)	**4900*** (100)	**4900*** (100)	4887.62 (36)
	0.5	4881.09 (0)	4881.17 (0)	4882.9 (0)	4899.41 (76)	4899.44 (78)	4893.97 (0)	4899.57 (81)	**4900*** (100)	**4900*** (100)	**4900*** (100)	4883.67 (0)
	0.6	4868.21 (0)	4868.74 (0)	4877.77 (0)	4895.3 (2)	4894.93 (0)	4888.52 (0)	4896.19 (9)	**4900*** (100)	**4900*** (100)	**4900*** (100)	4874.82 (0)
	0.7	4829.65 (0)	4831.94 (0)	4859.71 (0)	4872.41 (0)	4870.97 (0)	4864.38 (0)	4872.95 (0)	4894.58 (4)	(4)	**4896.33** (5)	4848.31 (0)
	0.8	4761.44 (0)	4737.73 (0)	4761.17 (0)	4766.67 (0)	4765.81 (0)	4763.93 (0)	4765.91 (0)	4698.78 (0)	(0)	4766.33 (0)	**4768.13** (0)

Table 3
Comparison of MPMA/Partial-MPMA with MMCOL [14] and Tr-ILS* [12] with a much relaxed time limit of 48h on the PLSE instances. Significantly best average results (t-test with p-value 0.001) are underlined. The percentage of instances for which the grid is completely filled is shown in parentheses.

Instance		Tr-ILS* (ext. time)			MMCOL (ext. time)			FastLSC (ext. time)			MPMA/Partial-MPMA		
n	r	f_{best}	f_{avg}	t(s)	f_{best}	f_{avg}	t(s)	f_{best}	f_{avg}	t(s)	f_{best}	f_{avg}	t(s)
50	0.3	**2500*** (100)	2500	1	**2500*** (100)	2500	0.22	**2500*** (100)	2500	0.12	**2500*** (100)	2500	142
	0.4	**2500*** (100)	2500	2	**2500*** (100)	2500	0.16	**2500*** (100)	2500	0.09	**2500*** (100)	2500	112
	0.5	**2500*** (100)	2500	2	**2500*** (100)	2500	0.31	**2500*** (100)	2500	0.13	**2500*** (100)	2500	89
	0.6	2499.63 (84)	2498.94	152	**2499.7** (85)	2499.7	17.55	(85)	2500	1.36	**2499.7** (85)	2499.7	489
	0.7	2473.53 (0)	2472.84	511	2479.13 (0)	2478.48	46996	(0)	-	-	**2484.38** (0)	2483.99	24970
	0.8	2394.34 (0)	2393.65	658	2378.15 (0)	2377.50	16268	(0)	-	-	**2394.58** (0)	2394.42	3916
60	0.3	**3600*** (100)	3600	2	**3600*** (100)	3600	0.69	**3600*** (100)	3600	0.24	**3600*** (100)	3600	326
	0.4	**3600*** (100)	3600	2	**3600*** (100)	3600	0.52	**3600*** (100)	3600	0.19	**3600*** (100)	3600	298
	0.5	**3600*** (100)	3600	17	**3600*** (100)	3600	0.67	**3600*** (100)	3600	0.24	**3600*** (100)	3600	214
	0.6	**3599.94** (97)	3599.25	69	**3599.94** (97)	3599.94	13.41	(97)	3600	2.46	**3599.94** (97)	3599.94	759
	0.7	3576.7 (0)	3576.01	1388	3590.22 (0)	3589.56	49279	(0)	-	-	**3593.52** (0)	3593.13	35658
	0.8	3478.92 (0)	3478.23	460	3457.07 (0)	3456.42	77979	(0)	-	-	**3480.03** (0)	3479.94	18141
70	0.3	**4900*** (100)	4900	3	**4900*** (100)	4900	0.90	**4900*** (100)	4900	0.70	**4900*** (100)	4900	721
	0.4	**4900*** (100)	4900	2	**4900*** (100)	4900	0.65	**4900*** (100)	4900	0.46	**4900*** (100)	4900	489
	0.5	4899.71 (92)	4899.22	18	**4900*** (100)	4900	1.51	**4900*** (100)	4900	0.56	**4900*** (100)	4900	349
	0.6	4899.98 (99)	4899.30	437	**4900*** (100)	4900	19.82	**4900*** (100)	4900	4.16	**4900*** (100)	4900	1210
	0.7	4880.10 (0)	4879.31	3245	4895.21 (5)	4894.54	55887	(5)	4900	10447	**4896.33** (5)	4895.93	46746
	0.8	4767.24 (0)	4766.33	2145	4736.70 (0)	4736.07	120862	(0)	-	-	**4768.13** (0)	4767.93	56670

On the other hand, using a very large population with a high diversity becomes critical when dealing with the most difficult instances such as those with $r \geq 0.7$. For these instances, MPMA obtains equal or better results compared to Tr-ILS* and MMCOL for all orders $n = 50, 60, 70$. Detailed results for the very difficult instances with $r = 0.7$ are displayed in Appendix A (Table A.1). Moreover, MPMA can optimally solve 25 of the 100 most challenging instances with $n = 70$ and $r = 0.7$ (cf. Table A.1).

It is difficult to compare the computation time between MPMA and the competitors, as MPMA takes advantage of a GPU while the other algorithms use a CPU. Therefore we compare MPMA and MMCOL in terms of number of iterations in order to observe whether the best results of MPMA come from the algorithm itself or from the parallelization. For this experiment, we do not consider FastLSC because it cannot solve any over-constrained PLSE instance for which the grid cannot be completely filled (indeed FastLSC is designed for the related LSC). As both MPMA and MMCOL use a one-move tabu search, the number of local search iterations is a suitable comparison criterion. We run MPMA and MMCOL with a maximum of 100 billions iterations of tabu search on the first ten instances of each of the most difficult (n, r) combinations with $n = 50, 60, 70$ and $r = 70, 80$. Each instance is independently solved 5 times. The detailed results are reported in Table 4, where we show for each instance and each algorithm (MMCOL, MPMA), the best result f_{best} over the 5 trials, the average result f_{avg} over these 5 trials, the average computation time in hours t(h) required to reach the best result and the average number of local search iterations nb_iter required to reach the best score. The best results are indicated in bold. According to the results, MPMA can achieve better or equal results for all instances with the same overall number of iterations. In addition, the use of a GPU reduces the time spent by the algorithm, because this important number of iterations can be performed in a shorter amount of time thanks to parallelization. This experiment confirms that the proposed MPMA algorithm dominates MMCOL.

In summary, MPMA and its Partial-MPMA variant for highly constrained instances (when $r > 0.7$) compete very favorably with the best performing PLSE methods in the literature, by reporting equal or better results on the 1800 benchmark instances. In Appendix B, we show that MPMA also performs extremely well on the special case of the Latin square completion problem, by attaining the optimal solutions for all the LSC benchmark instances.

5 Analysis of Important Factors in the Algorithm

We analyze the impacts of three important factors of the MPMA algorithm: (i) its very large population, (ii) the AUX crossover and (iii) the nearest neighbor matching strategy for parent selection. These experiments are based on the first ten hard instances with $n = 60$ and $r = 0.7$ of the PLSE.

5.1 Sensitivity to the Population Size

We first perform a sensitivity analysis of the algorithm with respect to the population size. For this, we perform the MPMA algorithm with p varying in the range $[10, 12288]$ to solve each of the ten instance 5 times under a time limit to 20 hours per run. Figure 8 displays the sensitivity of the average results to the population size p.

For the same time budget, the MPMA algorithm obtains better results with a larger size. When $p = 12288$, the algorithm always attains the best score over 10 runs. This can be explained by two reasons. First, due to the parallelization of the calculations on the GPUs, a large population improves the diversity of the population and helps the algorithm to perform a higher average global number of iterations at each run with the same time budget, which in turn increases the chance for the algorithm to attain high-quality solutions. Second, a large population increases the chance for each individual to find a closer but different nearest neighbor in the population for parent matching of the AUX crossover, which helps to generate promising offspring solutions.

Table 4
Comparison of MPMA with MMCOL [14] with a large number of iterations on the PLSE instances (maximum of 100×10^9 iterations). Significantly better average results (t-test with p-value 0.001) are underlined.

Instance	MMCOL (ext. nb iter.)				MPMA				Instance	MMCOL (ext. nb iter)				MPMA			
	f_{best}	f_{avg}	t(h)	nb_iter.	f_{best}	f_{avg}	t(h)	nb_iter		f_{best}	f_{avg}	t(h)	nb_iter	f_{best}	f_{avg}	t(h)	nb_iter
QC-50-70-1	2479	2478.2	168	82×10^9	**2485**	<u>2484.0</u>	4	36×10^9	QC-50-80-1	2380	2379.6	38	14×10^9	**2393**	<u>2392.8</u>	0.2	3×10^9
QC-50-70-2	2477	2476.6	45	24×10^9	**2482**	<u>2482.0</u>	2	20×10^9	QC-50-80-2	2377	2376.4	18	9×10^9	**2391**	<u>2391</u>	0.03	0.5×10^9
QC-50-70-3	2487	2486.6	150	81×10^9	**2490**	<u>2489.6</u>	5	44×10^9	QC-50-80-3	2381	2380.2	3	1×10^9	**2395**	<u>2395</u>	0.5	7×10^9
QC-50-70-4	2482	2481	54	29×10^9	**2487**	<u>2486.8</u>	3	28×10^9	QC-50-80-4	2386	2385.6	15	6×10^9	**2399**	<u>2399</u>	0.03	0.5×10^9
QC-50-70-5	2474	2474	139	68×10^9	**2482**	<u>2481.6</u>	10	94×10^9	QC-50-80-5	2377	2376.4	15	6×10^9	**2388**	<u>2388</u>	1.5	27×10^9
QC-50-70-6	2481	2480.2	28	14×10^9	**2485**	<u>2484.6</u>	5	44×10^9	QC-50-80-6	2378	2377.8	5	2×10^9	**2393**	<u>2393</u>	2	34×10^9
QC-50-70-7	2480	2479.6	186	92×10^9	**2485**	<u>2485.0</u>	6	57×10^9	QC-50-80-7	2387	2387	121	44×10^9	**2404**	<u>2403.8</u>	0.2	3×10^9
QC-50-70-8	2476	2475.4	98	49×10^9	**2483**	<u>2482.6</u>	6	62×10^9	QC-50-80-8	2367	2366.4	38	14×10^9	**2389**	<u>2389</u>	0.4	6×10^9
QC-50-70-9	2483	2482.4	45	24×10^9	**2486**	<u>2485.8</u>	8	84×10^9	QC-50-80-9	2378	2377.2	2	2×10^9	**2393**	<u>2393</u>	0.2	4×10^9
QC-50-70-10	2472	2471.4	26	10×10^9	**2480**	<u>2479.6</u>	9	75×10^9	QC-50-80-10	2362	2361	45	11×10^9	**2382**	<u>2382</u>	0.03	0.5×10^9
QC-60-70-1	3593	3592.8	121	51×10^9	**3594**	3593.8	6	39×10^9	QC-60-80-1	3448	3449.2	147	27×10^9	**3467**	<u>3467</u>	0.2	2×10^9
QC-60-70-2	3590	3589.4	71	36×10^9	**3594**	<u>3593.2</u>	9	59×10^9	QC-60-80-2	3453	3452.2	88	20×10^9	**3472**	<u>3472</u>	3	22×10^9
QC-60-70-3	3578	3577.2	160	55×10^9	**3583**	<u>3582.4</u>	14	91×10^9	QC-60-80-3	3454	3452.6	50	11×10^9	**3475**	<u>3474.8</u>	2	21×10^9
QC-60-70-4	3592	3591	164	58×10^9	**3595**	<u>3595</u>	11	72×10^9	QC-60-80-4	3464	3463.2	50	10×10^9	**3482**	<u>3482</u>	1	10×10^9
QC-60-70-5	3592	3591.2	267	94×10^9	**3594**	<u>3593.8</u>	7	42×10^9	QC-60-80-5	3471	3470.4	73	15×10^9	**3490**	<u>3489.6</u>	3	40×10^9
QC-60-70-6	3596	3595	287	99×10^9	**3598**	3597.0	10	64×10^9	QC-60-80-6	3450	3448	233	45×10^9	**3476**	<u>3476</u>	0.5	5×10^9
QC-60-70-7	3589	3588	162	57×10^9	**3591**	3590.8	8	52×10^9	QC-60-80-7	3459	3458.4	75	31×10^9	**3478**	<u>3478</u>	1.5	16×10^9
QC-60-70-8	3590	3589.2	241	85×10^9	**3593**	<u>3592.6</u>	9	59×10^9	QC-60-80-8	3464	3462.8	49	24×10^9	**3488**	<u>3488</u>	0.3	3×10^9
QC-60-70-9	3592	3592	94	36×10^9	**3595**	<u>3594.4</u>	13	85×10^9	QC-60-80-9	3448	3447.6	53	18×10^9	**3471**	<u>3471</u>	1.5	16×10^9
QC-60-70-10	3590	3589.2	235	65×10^9	**3592**	3591.4	8	51×10^9	QC-60-80-10	3457	3456.6	88	26×10^9	**3477**	<u>3477</u>	1.4	15×10^9
QC-70-70-1	4895	4894.8	47	15×10^9	**4897**	<u>4897.0</u>	18	84×10^9	QC-70-80-1	4729	4727.2	59	9×10^9	**4766**	<u>4764.8</u>	2	16×10^9
QC-70-70-2	4895	4895	151	51×10^9	**4896**	<u>4896.0</u>	9	42×10^9	QC-70-80-2	4738	4736.4	23	4×10^9	**4771**	<u>4768.6</u>	4	33×10^9
QC-70-70-3	4897	4896.6	125	48×10^9	4897	4897.0	11	51×10^9	QC-70-80-3	4721	4720.4	22	3×10^9	**4756**	<u>4755</u>	4	28×10^9
QC-70-70-4	4893	4892.4	69	27×10^9	**4894**	4893.8	17	79×10^9	QC-70-80-4	4735	4733.4	124	22×10^9	**4770**	<u>4767.8</u>	2	17×10^9
QC-70-70-5	4898	4898	177	91×10^9	4898	4897.6	8	38×10^9	QC-70-80-5	4748	4745.8	174	32×10^9	**4768**	<u>4768</u>	2	19×10^9
QC-70-70-6	4900	4898.6	63	29×10^9	4900	4899.2	7	32×10^9	QC-70-80-6	4747	4746.2	82	14×10^9	**4773**	<u>4772.8</u>	3	24×10^9
QC-70-70-7	4898	4897.8	146	71×10^9	4898	4897.6	19	89×10^9	QC-70-80-7	4746	4744	19	3×10^9	**4774**	<u>4774</u>	1	9×10^9
QC-70-70-8	4897	4896.6	189	92×10^9	**4898**	4898	9	43×10^9	QC-70-80-8	4745	4744.2	57	8×10^9	**4777**	<u>4775</u>	2	15×10^9
QC-70-70-9	4897	4896.2	85	52×10^9	**4898**	4896.8	16	81×10^9	QC-70-80-9	4730	4728.8	8	1×10^9	**4761**	<u>4760.4</u>	2	15×10^9
QC-70-70-10	4897	4896.4	93	46×10^9	4897	4896.8	7	31×10^9	QC-70-80-10	4750	4748.8	60	9×10^9	**4777**	<u>4776</u>	4	28×10^9

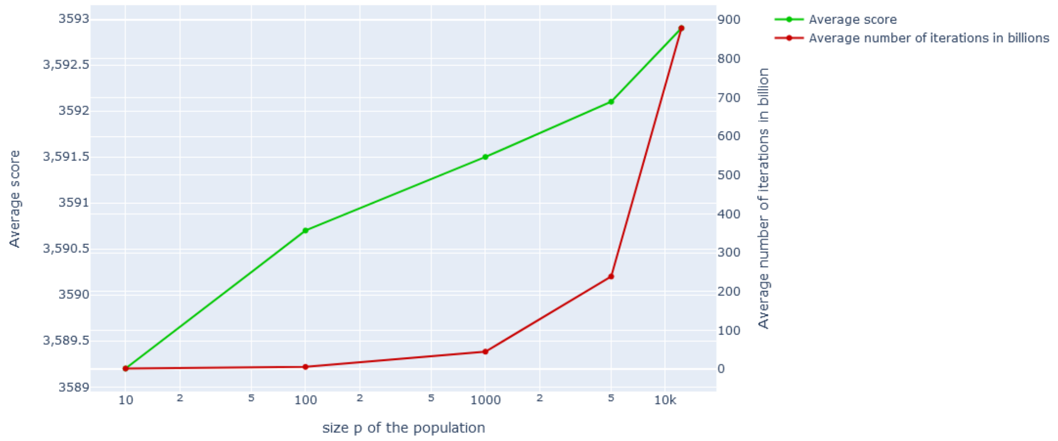

Fig. 8. Impact of the population size p on the performance of MPMA. Green curve corresponds to the average score and red curve to the average number of iterations in billions required to reach the best scores.

5.2 Impact of the Asymmetric Uniform Crossover

To study the impact of the asymmetric uniform crossover AUX on the MPMA algorithm, we compare it with four different variants of MPMA where the AUX crossover described in Section 3.5.2 is changed or disabled.

- The first variant is a baseline variant without crossover, so each offspring is an exact copy of its first parent.
- The greedy partition crossover GPX [28] is adapted for the Latin square problem: each color class of the offspring inherits the largest color class of the selected parent.
- The AUX crossover is replaced by the maximum approximate group based crossover MAGX of the MMCOL algorithm for the related Latin square completion problem [14].
- The AUX crossover is replaced by the uniform crossover (UX) which corresponds to AUX with p_{ij} being fixed to the value of 0.5.

Figure 9 shows the evolution of the best fitness values averaged over 5 runs for the same ten PLSE instances with $(n, r) = (60, 0.7)$ through the number of generations of each algorithm. One notices that the crossovers GPX and UX, which are the most disruptive, perform badly and are even outperformed by the variant without crossovers (blue line). This can be explained by the fact that the individuals are very distant in the population and rarely share large common features. Indeed, we experimentally observed that the average pairwise distance in the population is usually very large, around $0.7 \times |V|$.

The AUX and MAGX crossovers perform the best and dominate GPX and UX. Meanwhile, AUX dominates MAGX after 50 generations in average. The difference is statistically significant (confirmed by t-test with the p-value of 0.001). One reason to explain the advantage of AUX over MAGX is that with the AUX crossover, the offspring inherits more features from one parent than from the other parent. On the contrary, since MAGX is a symmetric crossover, crossing-over (S_i, S_j) and (S_j, S_i) lead to the same offspring, which results in less diversified offspring in the next generation.

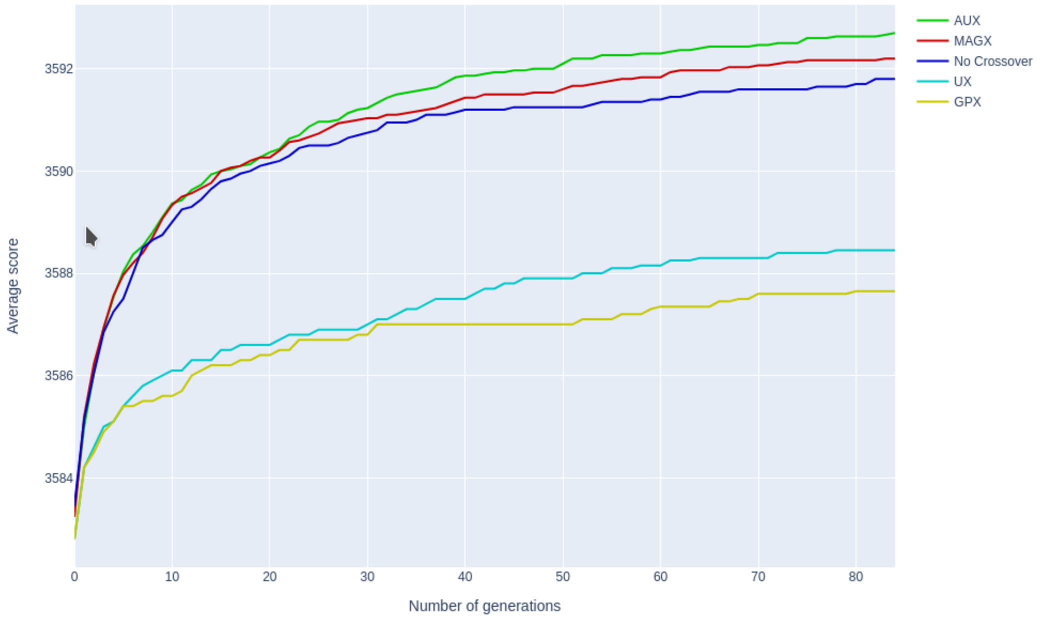

Fig. 9. Comparison of five different MPMA variants: No crossover (blue), GPX (yellow), MAGX (red), UX (light blue), AUX (green).

5.3 *Impact of the Crossover Matching Strategy*

To study the impact of the nearest neighbor matching strategy for the AUX crossover, we run a MPMA variant where this matching strategy is replaced by a random matching strategy: each individual as the first parent is cross-overed with another individual chosen randomly in the population.

Figure 10 shows the evolution of the best fitness values averaged over 5 runs for the same 10 first PLSE instances with $(n, r) = (60, 0.7)$ with respect to the number of generations of the algorithm. One notices that the matching strategy has an important impact on the performance. The dominance of the nearest neighbor matching strategy over the random matching becomes more and more evident after 10 generations. The difference is statistically significant (t-test with the p-value of 0.001). This is because two parents chosen randomly in the very large population share little information, leading to poor offspring whose quality can be hardly raised even after local optimization. The nearest neighbor strategy avoids this problem, as it does not destroy too much the color classes transmitted to the offspring, while preserving a certain level of diversity. This creates opportunities for the subsequent local search to explore new and interesting areas of the search space.

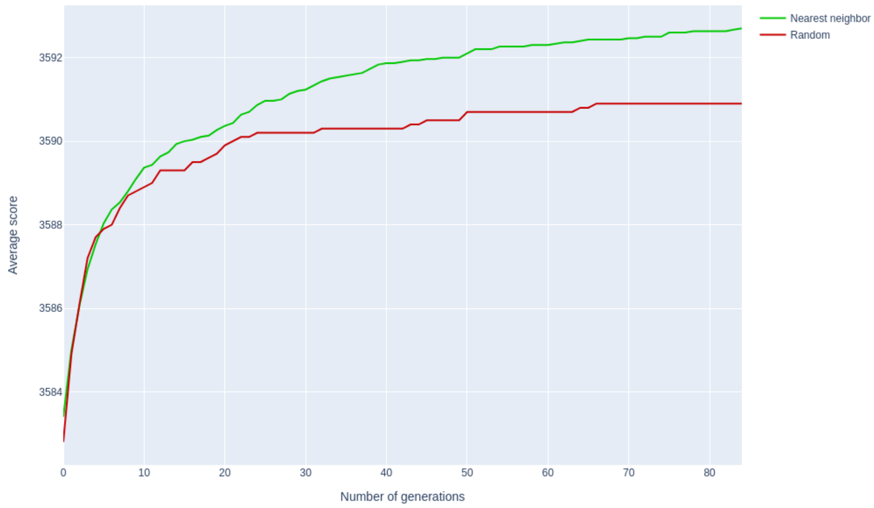

Fig. 10. Comparison of two parent matching strategies in MPMA: random matching (red) and nearest neighbor matching (green).

6 Conclusion

We presented a massively parallel population-based algorithm with a very large population and a practical implementation on GPUs to solve the partial Latin square extension problem as well as the special case of the Latin square completion problem. This approach highlights the interest of a very large population that enables massively parallel local optimization, offspring generations and distance calculations. The algorithm features a parameterized asymmetric crossover equipped with a dedicated parent matching strategy to build promising offspring, an effective parallel two-phase tabu search to improve new solutions and an original pool updating mechanism.

We performed extensive experiments to assess the proposed algorithm on the set of 1800 benchmark instances with various orders and ratios of pre-filled cells. The results showed that the algorithm obtains state-of-the-art results in average for all Latin square configurations (n, r). Furthermore, it definitely closed 25 challenging instances of order $n = 70$ and ratio $r = 0.7$. We investigated the impacts of key algorithmic components including the large population size and the parent matching strategy. This work demonstrates for the first time the high potential of GPU-based parallel computations for solving the challenging Latin square extension problem, by exploiting the formidable computing power offered by the GPUs and designing suitable search strategies.

The proposed algorithm can be used to solve relevant problems related to the PLSE. The availability of the source code of our algorithm will facilitate such applications. The design ideas of the algorithm can help to develop effective algorithms for other difficult combinatorial optimization problems. Future works could be carried out in particular to improve the parent matching strategy. For instance, it would be interesting to investigate strategies driven by a deep graph convolutional neural network in order to build the most promising offspring from appropriate parents.

CRediT author statement

Olivier Goudet: Conceptualization, Methodology, Software, Investigation, Writing - Original Draft. **Jin-Kao Hao:** Conceptualization, Methodology, Investigation, Writing - Original Draft.

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions which helped us to improve the paper. We would like to thank Dr. K. Haraguchi for sharing his Tr-ILS* code and the problem instances [12], Dr. Y. Jin for her assistance on their MMCOL code [14] and Dr. Y. Wang for sharing their FastLSC code [15]. This work was granted access to the HPC resources of IDRIS (Grant No. 2020-A0090611887, 2022-A0130611887) from GENCI.

References

[1] A. D. Keedwell, J. Dénes, Latin Squares and their Applications, Elsevier North Holland, 2015.

[2] D. Jakobovic, S. Picek, M. S. Martins, M. Wagner, Toward more efficient heuristic construction of boolean functions, Applied Soft Computing 107 (2021) 107327.

[3] R. A. Barry, P. A. Humblet, Latin routers, design and implementation, Journal of Lightwave Technology 11 (5/6) (1993) 891–899.

[4] C. J. Colbourn, The complexity of completing partial latin squares, Discrete Applied Mathematics 8 (1) (1984) 25–30.

[5] T. Evans, Embedding incomplete latin squares, The American Mathematical Monthly 67 (10) (1960) 958–961.

[6] F. E. Bennett, Quasigroup identities and mendelsohn designs, Canadian Journal of Mathematics 41 (2) (1989) 341–368.

[7] V. A. Artamonov, S. Chakrabarti, S. K. Pal, Characterization of polynomially complete quasigroups based on latin squares for cryptographic transformations, Discrete Applied Mathematics 200 (2016) 5–17.

[8] V. A. Artamonov, S. Chakrabarti, S. Gangopadhyay, S. Pal, On latin squares of polynomially complete quasigroups and quasigroups generated by shifts, Quasigroups and Related Systems 21 (2) (2013) 117–130.

[9] C. Gomes, D. Shmoys, Completing quasigroups or latin squares: A structured graph coloring problem, in: Proceedings of the Computational Symposium on Graph Coloring and Generalizations, 2002, pp. 22–39.

[10] C. Ansótegui, A. del Val, I. Dotú, C. Fernández, F. Manyà, Modeling choices in quasigroup completion: SAT vs. CSP, in: Proceedings of the 19th AAAI, AAAI Press, 2004, pp. 137–142.

[11] C. P. Gomes, R. G. Regis, D. B. Shmoys, An improved approximation algorithm for the partial latin square extension problem, Operations Research Letters 32 (5) (2004) 479–484.

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

[13] R. Lewis, A guide to graph colouring, Springer, 2015.

[14] Y. Jin, J.-K. Hao, Solving the latin square completion problem by memetic graph coloring, IEEE Transactions on Evolutionary Computation 23 (6) (2019) 1015–1028.

[15] S. Pan, Y. Wang, M. Yin, A fast local search algorithm for the latin square completion problem, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36, 2022, pp. 10327–10335.

[16] R. Banos, C. Gil, J. Reca, F. G. Montoya, A memetic algorithm applied to the design of water distribution networks, Applied Soft Computing 10 (1) (2010) 261–266.

[17] L. G. B. Ruíz, M. I. Capel, M. Pegalajar, Parallel memetic algorithm for training recurrent neural networks for the energy efficiency problem, Applied Soft Computing 76 (2019) 356–368.

[18] Z. Lü, J.-K. Hao, A memetic algorithm for graph coloring, European Journal of Operational Research 203 (1) (2010) 241–250.

[19] D. Donovan, The completion of partial latin squares, Australasian Journal of Combinatorics 22 (2000) 247–264.

[20] D. C. Porumbel, J.-K. Hao, P. Kuntz, An efficient algorithm for computing the distance between close partitions, Discrete Applied Mathematics 159 (1) (2011) 53–59.

[21] J.-K. Hao, Memetic algorithms in discrete optimization, in: Handbook of Memetic Algorithms, Springer, 2012, pp. 73–94.

[22] L. Moalic, A. Gondran, Variations on memetic algorithms for graph coloring problems, Journal of Heuristics 24 (1) (2018) 1–24.

[23] F. Neri, C. Cotta, P. Moscato (Eds.), Handbook of Memetic Algorithms, Vol. 379 of Studies in Computational Intelligence, Springer, 2012.

[24] W. Sun, J.-K. Hao, W. Wang, Q. Wu, Memetic search for the equitable coloring problem, Knowledge-Based Systems 188 (2020) 105000.

[25] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (4) (1987) 345–351.

[26] W. Sun, J.-K. Hao, X. Lai, Q. Wu, Adaptive feasible and infeasible tabu search for weighted vertex coloring, Information Sciences 466 (2018) 203–219.

[27] W. Wang, J.-K. Hao, Q. Wu, Tabu search with feasible and infeasible searches for equitable coloring, Engineering Applications of Artificial Intelligence 71 (2018) 1–14.

[28] P. Galinier, J.-K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (4) (1999) 379–397.

[29] E. Malaguti, M. Monaci, P. Toth, A metaheuristic approach for the vertex coloring problem, INFORMS Journal on Computing 20 (2) (2008) 302–316.

[30] G. Syswerda, Uniform crossover in genetic algorithms, in: J. D. Schaffer (Ed.), Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989, Morgan Kaufmann, 1989, pp. 2–9.

[31] I. Blöchliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme, Computers & Operations Research 35 (3) (2008) 960–975.

A Detailed Results for the Challenging PLSE Instances with $r = 0.7$

According to [12], instances with $r = 0.7$ are among the most challenging instances. Table A.1 presents the detailed results obtained by the MPMA algorithm on the three sets of 300 PLSE instance with $r = 0.7$ and $n = 50, 60, 70$.

Column 1 identifies the instances of each type (n, r). For each instance, we report the best PLSE score f_{best} (i.e., the largest number of filled cells) obtained over 5 runs with a maximum of 100 billions of tabu iterations, average score f_{avg} and average computation time t(s) in seconds to reach the best results. Bold values are the record-breaking results compared to the best-known results in the literature (including the best results obtained by running the codes of Tr-ILS* [12] and MMCOL [14] with the extended time limit of 48h). A star indicates an optimal value. The optimality is proved if (i) the number of filled cells reaches the upper bound $n^2 - l$ if $l \neq 1$ (cf. Section 2.2), or (ii) the number of filled cells is $n^2 - 2$ if $l = 1$ (cf. Theorem 6 in [19]). One observes that MPMA improves the best-known results for a large majority of the 300 instances and closes definitively 25 instances by reaching their optimal scores. Among these 25 optimal results, 14 were also achieved by MMCOL (starred non-bold values) with the extended time limit.

Table A.1
Detailed results of MPMA for the PLSE instance with $r = 0.7$

Id	PLSE-50-70			PLSE-60-70			PLSE-70-70		
	f_{best}	f_{avg}	t(s)	f_{best}	f_{avg}	t(s)	f_{best}	f_{avg}	t(s)
1	**2485**	2484.0	14634	**3594**	3593.8	21133	**4897**	4897.0	64501
2	**2482**	2482.0	7979	**3594**	3593.2	32775	**4896**	4896.0	31215
3	**2490**	2489.6	16640	**3583**	3582.4	50463	4897	4897.0	38822
4	**2487**	2486.8	11040	**3595**	3595.0	42003	**4894**	4893.8	60959
5	**2482**	2481.6	38581	**3594**	3593.8	23419	4898*	4897.6	29179
6	**2485**	2484.6	17336	**3598**	3597.0	35415	4900*	4899.2	24546
7	**2485**	2485.0	21093	**3591**	3590.8	29223	4898*	4897.6	68794
8	**2483**	2482.6	22696	**3593**	3592.6	31549	**4898**	4898.0	34353
9	**2486**	2485.8	29292	**3595**	3594.4	45923	**4898***	4896.8	56775
10	**2480**	2479.6	33675	**3592**	3591.4	27668	4897	4896.8	24600
11	**2488**	2488.0	10494	**3591**	3591.0	43547	**4895**	4895.0	41911
12	**2485**	2484.8	11099	**3595**	3595.0	29279	**4895**	4895.0	49769
13	**2483**	2482.0	35398	**3591**	3590.6	30085	4896	4896.0	38843
14	**2483**	2483.0	24327	3596	3594.8	12871	4900*	4900.0	49655
15	**2483**	2483.0	22104	**3598**	3597.6	42935	**4897***	4897.0	44162
16	**2484**	2484.0	24908	**3589**	3588.4	46321	**4895**	4894.4	51131
17	**2486**	2486.0	30868	**3589**	3588.2	44392	**4898***	4897.8	55798
18	**2489**	2488.6	43310	**3594**	3594.0	31095	**4896**	4895.2	45508
19	**2485**	2485.0	46223	**3592**	3591.6	34286	**4896**	4895.0	33190
20	**2490**	2490.0	66822	**3595**	3595.0	45880	4898*	4898.0	45274
21	**2483**	2482.8	10055	**3594**	3593.8	28887	**4896**	4895.8	43046
22	**2484**	2483.8	31473	**3594**	3593.6	36060	**4892**	4891.8	51100
23	**2485**	2485.0	59471	**3595**	3594.8	35139	4898*	4897.4	57851
24	**2488**	2487.2	39261	**3595**	3594.2	39917	**4896**	4895.6	62074
25	**2484**	2484.0	67246	**3595**	3595.0	24632	**4896**	4895.6	29744
26	**2483**	2482.4	11660	**3591**	3590.4	29959	**4896**	4896.0	49052
27	**2481**	2480.8	17530	**3596**	3595.6	21738	**4896**	4895.2	64276
28	**2484**	2484.0	57286	**3593**	3592.2	39360	**4895**	4895.0	30988
29	**2486**	2486.0	24712	**3594**	3593.8	36996	**4894**	4893.0	33507
30	**2485**	2484.4	32252	**3594**	3593.2	29165	**4894**	4893.8	67407
31	**2481**	2480.2	30863	**3596**	3595.8	28915	**4895**	4894.8	66784
32	**2481**	2480.8	26209	**3598**	3597.8	36874	**4898***	4898.0	63561
33	**2483**	2482.0	15300	**3594**	3593.8	50209	**4893**	4892.8	53578
34	**2484**	2483.6	17261	**3595**	3594.8	25322	**4896**	4896.0	29581
35	**2483**	2482.2	9258	**3594**	3593.6	48250	**4895**	4893.8	37046
36	**2484**	2483.8	39607	**3589**	3588.8	52302	**4896**	4896.0	30549
37	**2486**	2486.0	30891	**3592**	3591.8	36935	4898	4897.4	33335
38	**2479**	2479.0	27487	**3593**	3593.0	42965	**4896**	4895.8	46231
39	**2482**	2482.0	17885	**3592**	3592.0	40127	**4895**	4895.0	45687
40	**2486**	2485.8	25149	**3584**	3584.0	28671	**4897**	4897.0	39611
41	**2486**	2484.8	20498	**3593**	3593.0	44563	**4894**	4894.0	68759
42	**2485**	2484.2	29963	**3596**	3594.8	20232	4900*	4900.0	37155
43	**2486**	2485.6	22424	**3592**	3591.8	33863	**4897**	4896.4	65871
44	**2478**	2478.0	21238	**3596**	3595.2	39637	4900*	4900.0	24920
45	**2487**	2486.2	4387	**3594**	3593.2	53331	**4896**	4895.6	36570
46	**2486**	2485.2	8202	**3590**	3590.0	31509	**4895**	4894.4	66643
47	**2483**	2483.0	15598	**3596**	3596.0	50515	**4896**	4895.2	28201
48	**2485**	2485.0	12008	**3594**	3594.0	52813	**4898***	4898.0	45563
49	**2488**	2487.8	19546	**3592**	3592.0	30852	**4896**	4896.0	61217
50	**2487**	2486.2	36084	**3591**	3590.4	28170	**4892**	4891.2	63410
51	**2482**	2482.0	14454	**3597**	3596.8	27863	**4894**	4893.2	52307
52	**2483**	2482.8	3734	**3594**	3593.2	29788	**4894**	4893.6	47142
53	**2479**	2478.2	29808	**3590**	3590.0	34304	**4895**	4895.0	61063
54	**2482**	2482.0	31105	**3595**	3595.0	36915	4895	4894.8	49282
55	**2490**	2490.0	57119	**3593**	3592.8	43977	4898	4898.0	42535
56	**2486**	2485.2	16890	**3594**	3594.0	26958	**4897**	4896.6	40521
57	**2485**	2484.0	17693	**3596**	3595.6	22850	**4897**	4895.8	36423
58	**2484**	2483.6	22020	**3592**	3590.8	42025	**4895**	4895.0	50358
59	**2479**	2479.0	17566	**3597**	3597.0	35690	**4897**	4896.2	48762
60	**2485**	2483.8	12812	**3594**	3594.0	49378	4898*	4897.6	41952
61	**2488**	2487.6	32457	**3593**	3593.0	34521	**4896**	4895.6	40522
62	**2483**	2482.2	11236	**3595**	3595.0	36297	**4897**	4896.6	26971
63	**2484**	2483.8	49638	**3593**	3593.0	33612	4895	4894.0	36138
64	**2487**	2486.8	18411	**3589**	3589.0	45479	**4896**	4895.8	43970
65	**2483**	2483.0	14955	**3594**	3592.8	51097	**4895**	4894.2	64374
66	**2487**	2486.0	6173	**3594**	3593.8	32932	**4897**	4895.8	29134
67	**2492**	2491.4	13935	**3596**	3594.8	46629	**4896**	4895.2	39470
68	**2485**	2484.6	13185	**3591**	3591.0	46103	**4894**	4893.6	65397
69	**2480**	2478.8	61028	**3597**	3596.8	29333	**4896**	4895.4	33751
70	**2480**	2480.0	10097	**3596**	3595.0	21332	**4897**	4896.4	70332
71	**2485**	2484.8	14403	**3597**	3596.8	26326	**4898***	4896.8	37049
72	**2485**	2485.0	23233	**3593**	3593.0	41770	**4898***	4897.6	55525
73	**2481**	2481.0	13541	**3592**	3591.8	42388	**4897**	4896.2	41423
74	**2487**	2486.2	17062	**3591**	3590.4	41805	**4895**	4894.8	66514
75	**2486**	2485.8	6442	**3591**	3589.8	37734	**4893**	4892.6	27458
76	**2482**	2481.6	31480	**3596**	3595.6	32075	4895	4894.4	37566
77	**2484**	2484.0	30772	**3594**	3593.6	30518	**4898***	4897.8	71196
78	**2485**	2484.2	25027	**3594**	3593.8	37885	**4894**	4892.8	61918
79	**2486**	2485.4	11737	**3592**	3592.0	13140	**4895**	4894.8	55482
80	**2486**	2485.0	11477	**3591**	3590.6	17375	**4895**	4893.8	43852
81	**2484**	2483.6	25867	**3594**	3594.0	49588	**4898**	4898.0	34218
82	**2486**	2486.0	12979	**3596**	3595.2	49521	**4896**	4895.0	54607
83	**2484**	2483.8	45426	3591	3591.0	36875	**4897**	4896.8	53219
84	**2482**	2480.8	13416	**3597**	3596.8	39570	**4896**	4895.6	55558
85	**2486**	2485.2	18071	**3591**	3591.0	54486	**4895**	4894.2	51972
86	**2484**	2483.6	21323	**3591**	3590.8	32150	**4896**	4895.2	64547
87	**2482**	2481.8	7322	**3597**	3596.2	21235	**4898**	4897.8	58645
88	**2486**	2484.8	54905	**3595**	3594.0	31176	4898*	4897.6	55316
89	**2483**	2483.0	22200	**3596**	3595.2	42334	4898*	4898.0	69525
90	**2484**	2483.4	41303	**3591**	3590.6	28078	4898*	4897.8	33387
91	**2485**	2485.0	27282	**3595**	3594.8	37660	**4898***	4897.8	35507
92	**2480**	2478.8	64358	**3595**	3595.0	20591	**4898**	4898.0	44032
93	**2485**	2484.6	54895	**3595**	3594.4	34376	4898*	4897.4	30656
94	**2488**	2487.4	36221	**3593**	3592.6	27382	**4898***	4898.0	42838
95	**2485**	2484.2	34930	**3596**	3596.0	47333	**4895**	4894.0	15877
96	**2484**	2484.0	20119	**3588**	3587.4	26007	**4898***	4897.2	56506
97	**2483**	2482.2	12874	**3596**	3594.8	18748	**4895**	4894.6	52890
98	**2484**	2483.6	17232	**3595**	3593.8	39620	**4896**	4895.8	34437
99	**2487**	2487.0	11038	**3596**	3595.0	44668	4900*	4900.0	48432
100	**2481**	2481.0	6466	**3592**	3591.6	38165	**4895**	4895.0	41758

B Results on the Latin Square Completion Problem

Even if our MPMA algorithm is not designed for the Latin square completion (LSC) problem, the algorithm can be applied to the LSC because the latter can be considered as a special case of the partial Latin square extension problem. Two sets of LSC benchmark instances exist in the literature: 19 traditional instances from the COLOR03 competition [2] [9] and 1800 new instances [12]. These instances were built from complete Latin squares with some symbols removed. Thus these instances have the optimal score of n^2 (n is the order of the grid), i.e., their cells can be completely filled. Like the 1800 PLSE benchmark instances, these 1800 LCS instances have an order $n \in \{50, 60, 70\}$ and ratio $r \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, grouped to 18 subsets of 100 instances per (n, r) combination.

We ran the MPMA algorithm with a time limit of 3h with the parameters of Table 1 to solve the 1800 LCS instances. For the most difficult instances of the 19 traditional instances a time limit of 10 hours is required. The results on the set of 19 traditional instances (Table B.1) indicate that MPMA can solve all these instances with a perfect success rate. The best LSC algorithms MMCOL [14] and FastLSC [15] achieve a similar performance, but with a low success rate (1/30, 1/30 for MMCOL and 1/30, 1/30 for FastLSC) for two very difficult cases (qwhdec.order50.holes750.bal.1 and qwhdec.order60.holes1080.bal.1). However, MPMA requires a much higher computation time compared to MMCOL and FastLSC.

Table B.2 displays the results of the MPMA algorithm on the set of 1800 LCS instances compared to the state-of-the-art algorithms [12,14,15]. The results indicate that MPMA is able to solve all of these 1800 instances in the allotted time, matching the best LSC algorithms of [14,15].

[2] `http://mat.gsia.cmu.edu/COLOR03/`

Table B.1
Results of the MPMA algorithm on the set of 19 traditional LSC instances [9].

Instance			MMCOL		FastLSC		MPMA	
Name	n	r	SR	t(s)	SR	t(s)	SR	t(s)
qwhdec.order5.holes10.1	5	0.6	30/30	< 0.01	30/30	< 0.01	10/10	1.2
qwhdec.order18.holes120.1	18	0.63	30/30	< 0.01	30/30	< 0.01	10/10	1.9
qg.order30	30	0.0	30/30	0.04	30/30	0.02	10/10	22
qwhdec.order30.holes316.1	30	0.65	30/30	0.17	30/30	0.05	10/10	12
qwhdec.order30.holes320.1	30	0.64	30/30	1.37	30/30	0.13	10/10	4
qg.order40	40	0.0	30/30	0.17	30/30	0.09	10/10	55
qg.order60	60	0.0	30/30	1.22	30/30	0.65	10/10	526
qg.order100	100	0.0	30/30	17.5	30/30	10.66	10/10	3864
qwhdec.order33.holes381.bal.1	33	0.65	30/30	187.7	30/30	32.85	10/10	208
qwhdec.order35.holes405.1	35	0.67	30/30	16.5	30/30	5.30	10/10	56
qwhdec.order40.holes528.1	40	0.67	30/30	16.5	30/30	3.11	10/10	158
qwhdec.order60.holes1440.1	60	0.60	30/30	2.79	30/30	1.17	10/10	298
qwhdec.order60.holes1620.1	60	0.55	30/30	0.99	30/30	0.51	10/10	189
qwhdec.order70.holes2940.1	70	0.4	30/30	0.99	30/30	0.41	10/10	546
qwhdec.order70.holes2450.1	70	0.5	30/30	1.03	30/30	0.44	10/10	356
qwhdec.order50.holes825.bal.1	50	0.67	30/30	121	30/30	24.68	10/10	564
qwhdec.order50.holes750.bal.1	50	0.7	1/30	1444	1/30	448	10/10	10546
qwhdec.order60.holes1080.bal.1	60	0.7	1/30	2559	4/30	385	10/10	32484
qwhdec.order60.holes1152.bal.1	60	0.68	30/30	561	30/30	47.3	10/10	9556

Table B.2
Results of the MPMA algorithm on the 1800 new LSC instances [12] along with the results reported in the literature [12,14,15].

Instance		CPX-IP	CPX-CP	LSSOL	Tr-ILS*	MMCOL	FastLSC	MPMA
n	r	#Solved	#Solved	#Solved	#Solved	#Solved	#Solved	#Solved
50	30	9	94	10	100	100	100	100
	40	3	71	8	100	100	100	100
	50	0	12	6	100	100	100	100
	60	0	0	0	36	100	100	100
	70	0	0	0	0	100	100	100
	80	100	100	100	100	100	100	100
60	0.3	0	71	1	100	100	100	100
	0.4	0	22	0	100	100	100	100
	0.5	0	1	0	95	100	100	100
	0.6	0	0	0	23	100	100	100
	0.7	0	0	0	0	100	100	100
	0.8	100	100	99	99	100	100	100
70	0.3	0	34	0	99	100	100	100
	0.4	0	8	0	98	100	100	100
	0.5	0	0	0	84	100	100	100
	0.6	0	0	0	10	100	100	100
	0.7	0	0	0	0	100	100	100
	0.8	100	100	46	98	100	100	100