IPSJ SIG Technical Report

Vol.2015-AL-153 No.4
2015/6/12

A Collection of Efficient Local Searches for Partial Latin Square Extension Problem and Its Variants*

Kazuya Haraguchi[1,a)]

Abstract: The partial Latin square (PLS) extension problem asks for a largest extension of a given PLS. The problem is NP-hard and among fundamental problems in constraint optimization. Recently we proposed a collection of efficient local searches for the problem. The local search is based on conventional swap neighborhoods and on a novel type of neighborhood, Trellis-neighborhood. The iterated local searches with these neighborhoods outperform state-of-the-art optimization solvers by a wide margin. In this paper, we review this technique and extend it to variant problems such as Sudoku completion and finding orthogonal Latin squares.

Keywords: partial Latin square extension problem, maximum independent set problem, local search

1. Introduction

We address the *partial Latin square extension* (*PLSE*) problem. Let $n \geq 2$ be a natural number. Suppose that we are given an $n \times n$ grid of *cells*. A *partial Latin square* (*PLS*) is a partial assignment of *n symbols* to the cells so that the *Latin square condition* is satisfied. The Latin square condition requires that, in each row and in each column, every symbol should appear at most once. Given a PLS, the PLSE problem asks to fill as many empty cells with symbols as possible so that the Latin square condition continues to be satisfied. The problem is NP-hard [5], and was first studied by Kumar et al. [21]. It has been studied especially in the context of constant-ratio approximation algorithms [11], [14], [16], [21]. Currently the best approximation factor $3/4 - \varepsilon$ is achieved by a *local search* based algorithm [7], [8], [18].

In this paper, we review a collection of efficient local searches that we proposed recently [15]. Then we consider extending the local search to variant problems such as Sudoku completion and finding orthogonal Latin squares.

Being a well-known algorithmic framework, local search starts with an appropriate initial solution and then repeats moving to an improved solution as long as the *neighborhood* of the current solution contains one. The neighborhood is a set of solutions that are obtained by making "slight" modification on the current solution. Then local search is realized by repetition of a *neighborhood search algorithm*, which finds an improved solution in the neighborhood or concludes that no such solution exists.

For the modification on the current solution, we focus on *swap* operation. Given a solution PLS and non-negative integers p, q ($p < q$), (p, q)-*swap* is an operation of dropping exactly p symbols from the solution and then inserting at most q symbols to empty cells. The (p, q)-*neighborhood* is the set of all possible PLSs that are obtained by performing (p, q)-swap on the current solution.

In [15] we proposed (p, n^2)-neighborhood search algorithms that run in $O(n^{p+1})$ time for $p \in \{1, 2, 3\}$. Note that $q = n^2$ is the upper limit of the number of symbols that can be inserted in one swap operation. This means that, after dropping p symbols, we insert as many symbols to the solution as possible. We regard the proposed implementation as efficient; when $p = 1$, the time bound is linear with respect to the solution size. We also invented a novel type of swap operation, *Trellis-swap*, which is a generalization of (p, n^2)-swap with $p \leq 2$ and contains certain cases of $3 \leq p \leq n$. The proposed Trellis-neighborhood search algorithm runs in $O(n^{3.5})$ time. Computational experiments showed that, for randomly generated instances, the *iterated local search* (*ILS*) algorithm with Trellis-neighborhood is much more likely to deliver a better solution than not only ILS with (p, n^2)-neighborhoods but also state-of-the-art optimization softwares such as IP and CP solvers from IBM ILOG CPLEX [19] and a general heuristic solver from LocalSolver [22].

PLSE is a rather primitive problem and can be extended to various application problems; e.g., combinatorial design, scheduling, optical routers, and combinatorial puzzles [3], [6], [12]. We extend the local search to *PLSE on Colored Grid*, a generalization of Sudoku completion and finding orthogonal Latin squares (OLSs). Specifically we extend the (p, n^2)-neighborhood search algorithms with $p \leq 2$ to this problem. The running time is shown to be $O(n^{p+2})$.

The paper is organized as follows. We prepare terminologies and notations in Sect. 2. In Sect. 3 we review the collection of efficient local searches for the PLSE problem and present some experimental results. Then in Sect. 4, we consider extending the local search to PLSE on Colored Grid. Finally we conclude the paper in Sect. 5.

*This work is partially supported by JSPS KAKENHI Grant Number 25870661.

1 Faculty of Commerce, Otaru University of Commerce, Japan
a) haraguchi@res.otaru-uc.ac.jp

2. Preliminaries

We formulate the PLSE problem. Suppose an $n \times n$ grid of cells. We denote $[n] = \{1, 2, \ldots, n\}$. For any $i, j \in [n]$, we denote the cell in the row i and in the column j by (i, j). We consider a partial assignment of n symbols to the grid. The n symbols to be assigned are n integers in $[n]$. We represent a partial assignment by a set of triples, say $T \subseteq [n]^3$, such that the membership $(v_1, v_2, v_3) \in T$ indicates that the symbol v_3 is assigned to (v_1, v_2). To avoid a duplicate assignment, we assume that, for any two triples $v = (v_1, v_2, v_3)$ and $w = (w_1, w_2, w_3)$ in T $(v \neq w)$, $(v_1, v_2) \neq (w_1, w_2)$ holds. Thus $|T| \leq n^2$ holds.

For any two triples $v, w \in [n]^3$, we denote the Hamming distance between v and w by $\delta(v, w)$, i.e., $\delta(v, w) = |\{k \in [3] \mid v_k \neq w_k\}|$. We call a partial assignment $T \subseteq [n]^3$ a *PLS set* if, for any two triples $v, w \in T$ $(v \neq w)$, $\delta(v, w)$ is at least two. One easily sees that T is a PLS set iff it satisfies the Latin square condition. We say that two disjoint PLS sets S and S' are *compatible* if, for any $v \in S$ and $v' \in S'$, the distance $\delta(v, v')$ is at least two. Obviously the union of such S and S' is a PLS set. The PLSE problem is then formulated as follows; given a PLS set $L \subseteq [n]^3$, we are asked to construct a PLS set S of the maximum cardinality such that S and L are compatible.

In our local search, we treat the PLSE problem by reducing it to the *maximum independent set* (*MIS*) problem, a well-known NP-hard problem [9]. We then utilize sophisticated local search strategies on the MIS problem in the literature. The detail will be explained in the next section.

We review the MIS problem. An *undirected graph* (or simply a *graph*) $G = (V, E)$ consists of a set V of *vertices* and a set E of unordered pairs of vertices, where each element in E is called an *edge*. When two vertices are joined by an edge, we say that they are *adjacent*, or equivalently, that one vertex is a *neighbor* of the other. For any vertex, the number of its neighbors is called its *degree*. An *independent set* is a subset $V' \subseteq V$ of vertices such that no two vertices in V' are adjacent. Given a graph, the MIS problem asks for a largest independent set.

Suppose a graph $G^* = (V^*, E^*)$ with a vertex set $V^* = [n]^3$ and an edge set $E^* = \{(v, w) \in V^* \times V^* \mid \delta(v, w) = 1\}$. Each vertex $(v_1, v_2, v_3) \in V^*$ is a triple and regarded as a grid point in the 3D integral space, which is the intersection of three grid lines that are orthogonal to each other. Two vertices in G^* are joined by an edge iff there is a grid line that passes both of them. Any independent set in G^* should contain at most one vertex among those on a grid line. The following propositions are obvious.

Proposition 1 A set $S \subseteq [n]^3$ of triples is a PLS set iff S, as a vertex set, is an independent set in G^*.

Proposition 2 Two PLS sets L and S are compatible with each other iff $L \cup S$ is an independent set in G^*.

For a vertex $v \in V^*$, we denote by $N^*(v)$ the set of vertices adjacent to v, i.e., $N^*(v) = \{w \in V^* \mid \delta(v, w) = 1\}$. Clearly we have $|N^*(v)| = 3(n-1)$. For a triple set $L \subseteq [n]^3$, we denote by $N^*(L)$ the union $\bigcup_{v \in L} N^*(v)$ over L. A grid line is *in the direction* $d \in \{1, 2, 3\}$ if it is parallel to the axis d and perpendicular to the 2D plane that is generated by the other two axes. We denote by $\ell_{v,d}$ the grid line in the direction d that passes v.

Then we see that the PLSE problem on a PLS set $L \subseteq [n]^3$ is equivalent to the MIS problem on a subgraph $G = (V, E)$ of G^* induced by $V = V^* \setminus (L \cup N^*(L))$. We hereafter consider solving the PLSE problem by means of solving the MIS problem. For $v \in V$, we denote by $N(v) \subseteq N^*(v)$ the set of its neighbors in G. Since $|N(v)| \leq |N^*(v)| = 3(n-1)$ and $|V| = O(n^3)$, we have $|E| = O(n^4)$.

We call any independent set simply a *solution*. Given a solution $S \subseteq V$, we call any vertex $x \in S$ a *solution vertex* and any vertex $v \notin S$ a *non-solution vertex*. For a non-solution vertex v, we call any solution vertex in $N(v)$ a *solution neighbor of* v. We denote the set of solution neighbors by $N_S(v)$, i.e., $N_S(v) = N(v) \cap S$. Since v has at most one solution neighbor on one grid line and three grid lines pass v, we have $|N_S(v)| \leq 3$. We call the number $|N_S(v)|$ the *tightness of* v. When $|N_S(v)| = t$, we call v *t-tight*. In particular, a 0-tight vertex is called *free*. When x is a solution neighbor of a t-tight vertex v, we may say that v is a *t-tight neighbor of* x.

Given a solution S, *swap* operation in general drops a subset $D \subseteq S$ from S and then inserts a subset I into S so that $(S \setminus D) \cup I$ continues to be a solution. Dropping D from S makes certain vertices free: all vertices in D and non-solution vertices whose solution neighbors are completely contained in D. The inserted I should be an independent set among these free vertices. If there are D and I with $|D| < |I|$, then $(S \setminus D) \cup I$ is an improved solution.

For a generic σ-swap operation, the *σ-neighborhood* is the set of all possible solutions that are obtained by performing σ-swap on S. A solution is *σ-maximal* if its σ-neighborhood does not contain an improved solution. A *σ-neighborhood search algorithm* finds an improved solution in the σ-neighborhood of the input solution or decides that no such solution exists. Once a σ-neighborhood search algorithm is established, it is immediate to design a local search algorithm that computes a σ-maximal solution; starting with an appropriate initial solution, we repeat moving to an improved solution as long as the neighborhood search algorithm delivers one.

For two integers p, q with $0 \leq p < q$, *(p, q)-swap* refers to a swap operation with $|D| = p$ and $|I| \leq q$. We assume $q \leq n^2$ since the solution size is at most n^2. We call a solution *p-maximal* if it is (p, n^2)-maximal. In particular, we call a 0-maximal solution simply a *maximal* solution. Being p-maximal implies that S is also p'-maximal for any $p' < p$. Equivalently, if S is not p'-maximal, then it is not p-maximal either for any $p > p'$.

3. Local Search for PLSE Problem

In this section, we review the local search for the PLSE problem that we proposed in [15]. As mentioned in the last section, we deal with the PLSE problem by reducing it to the MIS problem. We borrow the data structure from the previous studies on local search for the MIS problem [2], [20]. Concerning (p, n^2)-neighborhood search ($p \in \{1, 2, 3\}$), our strategy is not merely a simple application of the previous approaches. We improve the efficiency by making use of the problem structure peculiar to the PLSE problem. Specifically, although a simple application may require $O(n^{p+3})$ time to conduct a $(p, p+1)$-neighborhood search, we achieved the following theorem.

Theorem 1 (Haraguchi [15]) Let $p \in \{1, 2, 3\}$. Given a solution S, we can find an improved solution in its (p, n^2)-neighborhood or conclude that it is p-maximal in $O(n^{p+1})$ time.

First we present the data structure in Sect. 3.1. We describe the (p, n^2)-neighborhood search algorithms in Sect. 3.2, only for $p = 1$ and 2. We then introduce the notion of Trellis-neighborhood and describe the neighborhood search algorithm in Sect. 3.3. In Sect. 3.4 we present some experimental results.

3.1 Data Structure

We store the graph $G = (V, E)$ by means of an ordering of vertices and a 3D array of vertices. The ordering is motivated by Andrade et al.'s local search [2] and used to scan vertices of a particular type (e.g., solution vertices, free vertices) in linear time with respect to their number. We introduce the 3D array to access the vertex in a specified coordinate if it exists or to decide that no such vertex exists in $O(1)$ time.

We denote an ordering of vertices by a bijection $\pi : V \to [|V|]$. In π, every solution vertex is ordered ahead of all the non-solution vertices. Among the non-solution vertices, every free vertex is ordered ahead of all the non-free vertices, and among the non-free vertices, every 1-tight vertex is ordered ahead of all 2-tight and 3-tight vertices. In each of the four sections (i.e., solution vertices, free vertices, 1-tight vertices and other non-free vertices), the vertices are ordered arbitrarily. We denote the inverse function by π^{-1}. We store π so that not only $\pi(v)$ ($v \in V$) but also $\pi^{-1}(i)$ ($i \in [|V|]$) can be accessed in $O(1)$ time. We also maintain parameters that represent the solution size, the number of free vertices and the number of 1-tight vertices, which we denote by $\#_{\mathrm{sol}}$, $\#_0$ and $\#_1$ respectively. We can access the head of each section in $O(1)$ time.

We denote the 3D $n \times n \times n$ array by C. For each triple $(v_1, v_2, v_3) \in [n]^3$, if $(v_1, v_2, v_3) \in V$, then we let $C[v_1][v_2][v_3]$ have a pointer to the vertex object of (v_1, v_2, v_3), and otherwise, we let it have a null pointer. The 3D array stores the edge set E implicitly. All the neighbors of (v_1, v_2, v_3) are among $C[v'_1][v_2][v_3]$'-s, $C[v_1][v'_2][v_3]$'-s and $C[v_1][v_2][v'_3]$'-s for every $v'_1, v'_2, v'_3 \in [n]$ such that $v'_1 \neq v_1$, $v'_2 \neq v_2$ and $v'_3 \neq v_3$.

In addition, for each solution vertex $x \in S$, we store a parameter $\lambda_d(x)$ ($d \in \{1, 2, 3\}$) that represents the number of its 1-tight neighbors in the direction d. For each non-solution vertex $v \notin S$, we store two parameters, $\tau(v)$ and $\rho_d(v)$. The former represents the tightness $|N_S(v)|$ of v and the latter represents the pointer to the solution neighbor of v in the direction d; when v has no such solution neighbor, we let $\rho_d(v)$ have a null pointer.

Clearly the size of the data structure is $O(n^3)$. We can construct it in $O(n^3)$ time, as preprocessing of local search. We show time complexities of some elementary operations.

Maximality check: We can check whether S is maximal or not in $O(1)$ time since it suffices to see whether $\#_0 = 0$ or $\#_0 > 0$.

Neighbor search: We can search all neighbors of a vertex in $O(n)$ time by using the 3D array C.

Drop: We can drop a solution vertex x from S in $O(n)$ time as follows. We exchange the orders between x and the last vertex in the solution vertex section. We decrease $\#_{\mathrm{sol}}$ by one and increase $\#_0$ by one; as a result, x falls into the free vertex section. The tightness $\tau(x)$ is set to zero and the pointers $\rho_d(x)$'-s ($d \in \{1, 2, 3\}$) are assigned null pointers. For every neighbor $v \in N(x)$ in each direction d, we release its pointer $\rho_d(x)$ to x since x is no longer a solution vertex, and decrease the tightness $\tau(v)$ by one.

- If $\tau(v)$ is decreased to zero, then v is now free. To put v in the free vertex section, we exchange the orders between v and the head vertex in the 1-tight vertex section, and increase $\#_0$ by one and decrease $\#_1$ by one.
- If $\tau(v)$ is decreased to one, then v is now 1-tight. To put v in the 1-tight vertex section, we exchange the orders between v and the head vertex in the last vertex section, and increase $\#_1$ by one. Furthermore, v has a unique solution neighbor, say y. We increase the number $\lambda_d(y)$ by one, where d is the direction of the grid line that passes both v and y.

Insertion: We can insert a free vertex into S in $O(n)$ time in a manner analogous to the drop operation.

3.2 (p, n^2)-Neighborhood Search Algorithms

The (p, n^2)-neighborhood search algorithms have the similar structure. In principle, we need to search all subsets D'-s of S of size p. We do this by means of scanning "trigger" vertices. When $p = 1$ (resp., 2), each solution vertex (resp., 2-tight vertex) serves as a trigger vertex. For each trigger vertex, we generate a subset D of size p somehow in $O(1)$ time. Let F denote the set of vertices free from $S \setminus D$. We then compute the MIS size of the subgraph induced by F. We may call it "the MIS size of F" for simplicity and denote it by $\#_{\mathrm{MIS}}$ when F is clear from the context. The point is that we compute $\#_{\mathrm{MIS}}$ without finding an MIS itself. Surprisingly we can compute $\#_{\mathrm{MIS}}$ in $O(1)$ time. Only when $\#_{\mathrm{MIS}} > p$, we search for an MIS to be inserted, denoted by I, and thereby obtain an improved solution $(S \setminus D) \cup I$. The search for I requires $O(n)$ time.

The (p, n^2)-neighborhood search algorithms are established based on this top-level strategy. Below we describe the algorithms for $p = 1$ and 2.

3.2.1 Case of $p = 1$

Let S be a maximal solution. The $(1, n^2)$-neighborhood contains an improved solution iff there are $x \in S$ and $u, v \notin S$ such that $(S \setminus \{x\}) \cup \{u, v\}$ is a solution. It is clear that u and v should be neighbors of x. They are 1-tight, and their unique solution neighbor is x. The u and v should not be adjacent, which implies that u and v are not on the same grid line. We define $\Lambda(x)$ as a subset of the direction indices in which x has a 1-tight neighbor, i.e., $\Lambda(x) = \{d \in \{1, 2, 3\} \mid \lambda_d(x) > 0\}$. Then $\#_{\mathrm{MIS}}$, the largest number of vertices that can be inserted into $S \setminus \{x\}$ simultaneously, is given by the cardinality $|\Lambda(x)|$. Therefore, an improved solution exists iff there is $x \in S$ with $|\Lambda(x)| > 1$.

Given a solution S, we can find an improved solution in the $(1, n^2)$-neighborhood or conclude that it is 1-maximal in $O(n^2)$ time as follows; First, if S is not maximal, we obtain an improved solution by inserting any free vertex into S, which requires $O(n)$ time. Then we assume S to be maximal and thus utilize the above argument. We search all solution vertices by sweeping the first section of the vertex ordering. There are at most n^2 solution vertices. For each solution vertex x, the number $|\Lambda(x)|$ can be com-

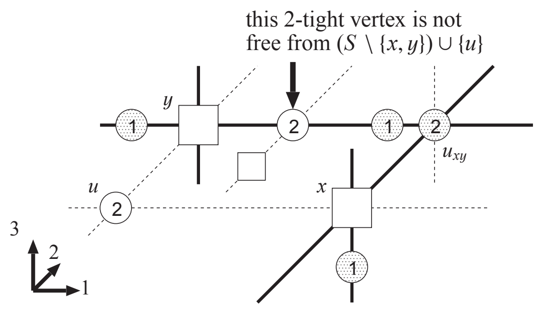
this 2-tight vertex is not free from $(S \setminus \{x, y\}) \cup \{u\}$
y
u
x
u_{xy}
1
2
1
2
2
1
3
2
1

Fig. 1 An illustration of the case such that $(2, n^2)$-swap can be made: A square □ indicates a solution vertex and a circle ○ indicates a non-solution vertex, where the number in ○ indicates the tightness. Shaded vertices are free from $(S \setminus \{x, y\}) \cup \{u\}$.

puted in $O(1)$ time. If x with $|\Lambda(x)| > 1$ is found, we can determine the 1-tight vertices to be inserted in $O(n)$ time by searching each grid line $\ell_{x,d}$ with $\lambda_d(x) > 0$. The number of inserted 1-tight vertices is at most 3. Dropping x from S and inserting the 1-tight vertices into $S \setminus \{x\}$ take $O(n)$ time.

3.2.2 Case of $p = 2$

Let S be a 1-maximal solution. The $(2, n^2)$-neighborhood contains an improved solution iff there exist $x, y \in S$ and $u, v, w \notin S$ such that $(S \setminus \{x, y\}) \cup \{u, v, w\}$ is a solution. These vertices should satisfy Lemmas 1 to 4 in [2], which are conditions established for the general MIS problem. According to the conditions, each vertex in $\{u, v, w\}$ is either 1-tight or 2-tight, and at least one of them is 2-tight. The unique solution neighbor of a 1-tight vertex is either x or y, and the two solution neighbors of a 2-tight vertex are x and y.

For any 2-tight vertex u and its solution neighbors x and y, let F denote the set of vertices free from $(S \setminus \{x, y\}) \cup \{u\}$. We would like to know the MIS size of F. To observe how vertices in F are distributed, see Fig. 1. In the figure, we permute the coordinates so that x (resp., y) is the u'-s solution neighbor in the direction 1 (resp., 2). Indicated by shade, vertices in F are among the four solid grid lines, that is, $\ell_{x,2}$, $\ell_{x,3}$, $\ell_{y,1}$ and $\ell_{y,3}$. Let us denote the intersection point of $\ell_{x,2}$ and $\ell_{y,1}$ by $u_{xy} = (u_{xy,1}, u_{xy,2}, u_{xy,3})$. Formally, it is defined as $u_{xy,1} = x_1$, $u_{xy,2} = y_2$, and $u_{xy,3} = u_3$, where the last one is also equal to x_3 and y_3. Then F contains all 1-tight vertices on the four grid lines, and also contains the vertex u_{xy} only if it exists and is 2-tight.

The MIS size of F can be computed in $O(1)$ time without computing F explicitly; one can show that the size is either $|\Lambda(x) \setminus \{1\}| + |\Lambda(y) \setminus \{2\}|$ or $|\Lambda(x) \setminus \{1\}| + |\Lambda(y) \setminus \{2\}| + 1$, and it is the latter iff the vertex u_{xy} exists, it is 2-tight, and $\lambda_2(x) = \lambda_1(y) = 0$.

Then given a solution S, we can find an improved solution in the $(2, n^2)$-neighborhood or conclude that it is 2-maximal in $O(n^3)$ time as follows; We assume S to be 1-maximal since, if not so, we obtain an improved solution in $O(n^2)$ time. We search all 2-tight vertices by sweeping the last section of the vertex ordering, and their number is at most $|V \setminus S| \leq |V| \leq n^3$. The solution vertices to be dropped are the solution neighbors of u, which we denote by x and y. We can recognize x and y in $O(1)$ time by tracing the pointers $\rho_d(u)$'-s. Let F be the set of vertices free from $S \setminus \{x, y\} \cup \{u\}$. We can compute the MIS size of F in $O(1)$ time. If the MIS size is no less than 2, there exists an improved solution. An MIS of F is decided in $O(n)$ time by searching the four grid lines. The total number of inserted vertices is at most 4. Drop and insertion requires $O(n)$ time.

3.3 Trellis-Neighborhood Search Algorithm

In this section, we introduce a novel type of neighborhood, Trellis-neighborhood, and describe a neighborhood search algorithm that runs in $O(n^{3.5})$ time.

Trellis-neighborhood is a generalization of (p, n^2)-neighborhood with $p \leq 2$. Let $d \in [3]$ and $k \in [n]$ be any integers. We regard a vertex v as a 3D integral point $v = (v_1, v_2, v_3)$. Cutting the $n \times n \times n$ 3D integral cube by a 2D plane $v_d = k$, we have a 2D facet. We call this facet the (d, k)-*facet*. There are possibly some vertices on the (d, k)-facet. In Trellis-swap, the set D of solution vertices to be dropped is any subset of solution vertices on the facet. We define the *Trellis-neighborhood of a solution* S as the set of all solutions that can be obtained by dropping such D from S and then inserting an independent set I into $S \setminus D$. Observe that (p, n^2)-neighborhood with $p \leq 2$ is a special case of Trellis-neighborhood in the sense that the cardinality $|D|$ is restricted to p.

We claim that, even when D is maximal (i.e., D is the set of all solution vertices on a (d, k)-facet), a largest I can be computed efficiently in $O(n^{2.5})$ time since the problem of finding a largest I is reduced to the maximum bipartite matching problem. In what follows, we restrict ourselves to such D. Then $|D|$ is at most n and there are at most $3n$ different D'-s. Let us define the (d, k)-*Trellis*, a certain subgraph of G.

Definition 1 Suppose that we are given a solution S. For any $d \in [3]$ and $k \in [n]$, let $D \subseteq S$ be a subset $D = \{(v_1, v_2, v_3) \in S \mid v_d = k\}$. We denote by F_1 (resp., F_2) the set of all 1-tight (resp., 2-tight) vertices such that the unique solution neighbor is contained in D (resp., both of the solution neighbors are contained in D). We define the (d, k)-*trellis* as the subgraph of G induced by $D \cup F_1 \cup F_2$.

We may call the (d, k)-facet and the (d, k)-trellis simply the facet and the trellis respectively when d and k are clear from the context.

We show an example of trellis in Fig. 2. The D consists of four solution vertices on the 2D facet, i.e., $D = \{x, y, z, z'\}$. All vertices in the trellis are indicated by shade; F_1 is the set of five 1-tight vertices, and F_2 is the set of three 2-tight vertices. We see that all vertices in $D \cup F_2$ are on the 2D facet. Two vertices in F_1 are also on the same facet, while the three other vertices in F_1 are out of the facet like a hanging vine.

Note that dropping D from S makes all vertices in the trellis free. Then any independent set in the trellis serves as I.

Lemma 1 Given $d \in [3]$ and $k \in [n]$, we can compute an MIS of the (d, k)-trellis in $O(n^{2.5})$ time.

Proof: We explain how to compute an MIS of the (d, k)-trellis. Let us partition F_1 into $F_1 = F_1' \cup F_1''$ so that F_1' (resp., F_1'') is the subset of vertices on (resp., out of) the (d, k)-facet. The vertex set F_1'' induces a subgraph that consists of cliques, each of which is formed by 1-tight vertices on a grid line perpendicular to the facet; in Fig. 2, ℓ and ℓ' are among such grid lines. Among MISs of the trellis, there is one such that a solution vertex is chosen

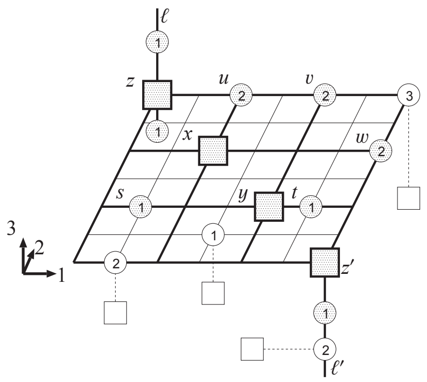

Fig. 2 An example of trellis: Four bold squares on the same 2D facet are the solution vertices to be dropped. Vertices in the trellis are indicated by shade.

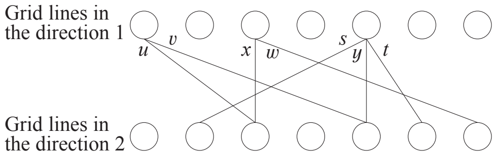

Fig. 3 The bipartite graph in the proof of Lemma 1: Any matching is an independent set of the trellis. $\{x, y\}$ is a matching that consists of vertices in the current solution, and we see a larger matching such as $\{s, v, w\}$.

from every clique. Intending such an MIS, we ignore the vertices in F''_1 and their unique solution neighbors. Let $D' \subseteq D$ be a subset such that $D'' = \bigcup_{u \in F''_1} N_S(u)$; in Fig. 2, $D'' = \{z, z'\}$. We can no longer choose the vertices in D''. Let $D' = D \setminus D''$.

The remaining task is to compute an MIS of $D' \cup F'_1 \cup F_2$. All the vertices in $D' \cup F'_1 \cup F_2$ are on the (d, k)-facet. We should choose as many vertices as possible so that, from each of $2n$ grid lines on the facet, a vertex is chosen at most once. It is the problem of finding a maximum matching in a bipartite graph like Fig. 3; a vertex is associated with a grid line on the facet, and the bipartition of vertices is determined by the directions of grid lines. An edge joins two vertices whenever the trellis has a vertex on the intersecting point of the corresponding two grid lines.

An MIS of the trellis is given by the union of D'' and a maximum matching M. It takes $O(n^2)$ time to recognize the vertex sets D, F_1 and F_2 and partitions $D = D' \cup D''$ and $F_1 = F'_1 \cup F''_1$, and then to construct the bipartite graph since the bipartite graph has $2n$ vertices and $O(n^2)$ edges. We need $O(n^{2.5})$ time to compute M [17]. □

Theorem 2 (Haraguchi [15]) Given a solution S, we can find an improved solution in the Trellis-neighborhood or conclude that no such solution exists in $O(n^{3.5})$ time.

Proof: There are $3n$ facets and thus $3n$ trellises. For a (d, k)-trellis, let D be the maximal subset of solution vertices on the trellis. We can identify whether there is an independent set I with $|D| < |I|$ or not in $O(n^{2.5})$ time from Lemma 1. When such D and I are found, we have an improved solution $(S \setminus D) \cup I$ by dropping D from S and then by inserting I into $S \setminus D$. This requires $O(n^2)$ time since $|D| \leq n$ and $|I| \leq 2n$. □

3.4 Experimental Results

We design iterated local search (ILS) algorithms [10], [13] that utilize the neighborhood search algorithms described so far. We then compare them with state-of-the-art optimization softwares in terms of solution quality; Our ILS algorithms outperform the competitors.

The ILS algorithm iterates local searches until the computation time exceeds a given time limit (which is set to 10 seconds), and then outputs the incumbent solution S^*, i.e., the best solution searched so far. Each local search begins with an initial solution and repeats a σ-neighborhood search algorithm until it finds a σ-maximal solution, say S. If S is not worse than the current S^* (i.e., $|S| \geq |S^*|$), then S^* is updated to S. The initial solution S_0 of the next local search is generated by "kicking" S^*, where we write the detail in our future papers. We consider four variants of ILS algorithms: 1-ILS, 2-ILS, 3-ILS and Tr-ILS. As the name goes, 1-ILS is the ILS algorithm with $(1, n^2)$-neighborhood. The others are analogous.

For competitors, we employ two exact solvers and one heuristic solver. For the former, we employ the optimization solver for integer programming model (CPX-IP) and the one for constraint optimization model (CPX-CP) from IBM ILOG CPLEX (ver. 12.6) [19]. It is easy to formulate the PLSE problem by these models (e.g., see [12]). For the latter, we employ LocalSolver (ver. 4.5) [22] (LSSOL), which is a general heuristic solver based on local search. All the parameters are set to default values except that, in CPX-CP, `DefaultInferenceLevel` and `AllDiff InferenceLevel` are set to `extended`. We set the time limit of all the competitors to 30 seconds (which is three times the time limit of ILS algorithms).

Benchmark instances are random PLSs. We generate the instances by utilizing the scheme called "quasigroup completion" (QC) that is well-known in the literature [4], [12]. Note that a PLS is parametrized by the order n and the ratio $r \in [0, 1]$ of pre-assigned symbols over the $n \times n$ grid. Starting from an empty assignment, QC generates a PLS by assigning a symbol to an empty cell randomly until $\lfloor n^2 r \rfloor$ symbols are assigned. Note that a QC instance does not necessarily admit a complete Latin square as an optimal solution. One can download some of the used instances from the author's website (`http://puzzle.haraguchi-s.otaru-uc.ac.jp/PLSE/`).

Let us mention what kind of instance is "hard" in general. Of course an instance becomes harder when n is larger. Then we set the grid length n to 50, 60 and 70, which are relatively large compared with previous studies (e.g., [12]). For a fixed n, the problem has easy-hard-easy phase transition. Then we regard instances with an intermediate r "hard".

All the experiments are conducted on a workstation that carries Intel® Core™ i7-4770 Processor (up to 3.90GHz by means of Turbo Boost Technology) and 8GB main memory. The installed OS is Ubuntu 14.04.1.

We show how the ILS algorithms and the competitors improve the initial solution S_0 in Table 1. The S_0 is generated by a constructive algorithm named G5 in [1], which is a "look-ahead" minimum-degree greedy algorithm. For each pair (n, r), a number in the 3rd column is the average of $|L| + |S_0|$ (i.e., the given PLS size $|L| = \lfloor n^2 r \rfloor$ plus the initial solution size) and a number in the 4th to 10th columns is the average of the improved size over 100

QC instances. A bold number indicates the largest improvement among all.

Obviously the ILS algorithms outperform the competitors in many (n, r)'-s although the time limit is much shorter. We claim that Tr-ILS should be the best among the four ILS algorithms. Clearly 3-ILS is inferior to others; this must be because the $(3, n^2)$-neighborhood search algorithm takes much longer time than others. The remaining three algorithms seem to be competitive, but Tr-ILS ranks first or second most frequently.

Concerning the competitors, CPX-CP performs well for under-constrained "easy" instances (i.e., $r \le 0.4$), whereas CPX-IP does well for over-constrained "easy" instances (i.e., $r \ge 0.7$). LSSOL is relatively good for all r'-s and outstanding especially for "hard" instances with $0.5 \le r \le 0.7$.

4. Extension of Local Search for PLSE on Colored Grid

Now we extend the local search to a variant of the PLSE problem, *PLSE on Colored Grid* (*PLSE-CG*). In this problem, the $n \times n$ grid is assumed to be colored, that is, each cell is assigned one of n colors in a given palette, say $c_1, \ldots, c_n$. We denote the color of cell (i, j) by $\mathrm{col}(i, j)$. We say that a PLS satisfies the *color condition* if, among the cells having the same color, each symbol appears at most once. Then given a colored grid and a PLS L that satisfies the color condition, the PLSE-CG problem asks for a largest extension of L so that not only the Latin square condition but also the color condition continues to be satisfied.

The PLSE-CG problem is a generalization of (maximization versions of) Sudoku completion and finding orthogonal Latin squares (OLSs). In Sudoku completion, the order n is assumed to be a square number $n = m^2$. Then it is regarded as a special case of the PLSE-CG problem such that, in every $m \times m$ subgrid, all n cells are assigned the same color, and any two cells in different subgrids are assigned different colors from each other. Two Latin squares A, B are called *orthogonal* if every ordered pair $(a, b) \in [n]^2$ of symbols appears exactly once in $\bigcup_{(i,j)\in[n]^2}\{(a, b) \mid (i, j, a) \in A \text{ and } (i, j, b) \in B\}$. To find a Latin square that is orthogonal to a given A, we have only to solve the PLSE-CG problem such that $\mathrm{col}(i, j) = c_k$ whenever $(i, j, k) \in A$.

Let $G^{**} = (V^*, E^* \cup E^\dagger)$ be a graph such that the new edge set $E^\dagger$ is added to the graph G^*. This edge set is defined as;

$$E^\dagger = \{(u, v) \in V^* \times V^* \mid \mathrm{col}(u_1, u_2) = \mathrm{col}(v_1, v_2),\ u_3 = v_3\}.$$

Then vertices (u_1, u_2, u_3)'-s with the same color $\mathrm{col}(u_1, u_2)$ and the same u_3 form a clique. We denote by $N^{**}(v)$ the set of v'-s neighbors in G^{**}. The PLSE-CG problem is reduced to the MIS problem on the subgraph of G^{**} induced by $V = V^* \setminus (L \cup N^{**}(L))$. For convenience, we represent the adjacency given by $E^\dagger$ by means of a grid line; that is, we regard that u and v are on the grid line "in the direction 4" if $(u, v) \in E^\dagger$. We connect $C[i][j][k]$'-s by double-links to represent a grid line in the direction 4. The tightness of a non-solution node is now at most 4. In addition to 3 directions, we maintain the number $\lambda_4(x)$ for any solution vertex x and the pointer $\rho_4(u)$ for any non-solution vertex u. It is easy to see that we can do elementary operations in Sect. 3.1 within the same time bound.

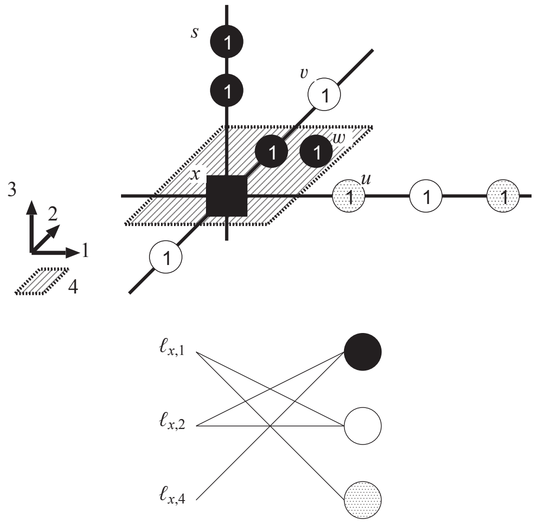

Fig. 4 An illustrative example for the proof of Theorem 3: (upper) distribution of vertices in F and (lower) the bipartite graph for this situation

We extend the (p, n^2)-neighborhood search algorithm with $p \le 2$ to the PLSE-CG problem. First we consider the case of $p = 1$. Let $x \in S$ be any solution vertex, and F denote the set of vertices free from $S \setminus \{x\}$. Recall that F consists of 1-tight neighbors of x. In contrast to the primitive PLSE problem, we cannot pick up a 1-tight vertex from each grid line independently. It is possible that, for example, u on $\ell_{x,1}$ and v on $\ell_{x,2}$ are on the same grid line $\ell_{x,4}$. In such a case, u and v cannot be inserted into $S \setminus \{x\}$ simultaneously.

Theorem 3 In the PLSE-CG problem, given a solution S, we can find an improved solution in its $(1, n^2)$-neighborhood or conclude that it is 1-maximal in $O(n^3)$ time.

Proof: It suffices to show that the MIS of F can be computed in $O(n)$ time. We construct a bipartite graph that consists of three vertices corresponding to solid grid lines $\ell_{x,1}$, $\ell_{x,2}$ and $\ell_{x,4}$ and n vertices corresponding to colors in a given palette. Let F' denote the subset of vertices in F that are on the grid line $\ell_{x,3}$. We define the edge set as follows.

- If there is $u \in F \setminus F'$ that is on $\ell_{x,1}$, then we join $\ell_{x,1}$ and $\mathrm{col}(u_1, u_2)$ by an edge.
- If there is $u \in F \setminus F'$ that is on $\ell_{x,2}$, then we join $\ell_{x,2}$ and $\mathrm{col}(u_1, u_2)$ by an edge.
- If there is $u \in F \setminus F'$ that is neither on $\ell_{x,1}$ nor on $\ell_{x,2}$ but is on $\ell_{x,4}$, then we join $\ell_{x,4}$ and $\mathrm{col}(u_1, u_2)$ by an edge.

See Fig. 4 for example. We claim that a matching in this bipartite graph corresponds to an independent set of F. Since there are only 3 vertices on one side of bipartition and are $O(n)$ edges, it takes $O(n)$ time to compute a maximum matching. Finally, from F', we can add any 1-tight vertex on $\ell_{x,3}$ to the independent set if one exists. □

In the example of Fig. 4, an independent set $\{u, v, w\}$ may be chosen by a maximum matching algorithm, and s on the grid line $\ell_{x,3}$ may be appended into the independent set. Then we have an improved solution $(S \setminus \{x\}) \cup \{u, v, w, s\}$.

Theorem 4 In the PLSE-CG problem, given a solution S, we can find an improved solution in its $(2, n^2)$-neighborhood or conclude that it is 2-maximal in $O(n^4)$ time.

Table 1 Improved sizes brought by the ILS algorithms and the competitors

n	r	$\lvert L\rvert + \lvert S_0\rvert$	Tr-ILS	1-ILS	2-ILS	3-ILS	CPX-IP	CPX-CP	LSSOL
50	0.3	2496.03	**3.97**	**3.97**	3.95	3.93	0.00	3.84	0.32
	0.4	2493.78	**6.22**	6.20	**6.22**	6.08	0.00	4.24	0.87
	0.5	2488.52	**11.48**	11.37	11.43	10.73	0.00	1.40	4.44
	0.6	2476.21	**20.97**	20.02	20.09	18.46	0.00	2.66	13.00
	0.7	2442.21	**27.86**	27.26	27.57	25.56	4.19	8.83	21.24
	0.8	2382.07	12.07	12.07	12.04	12.02	**12.51**	6.03	11.60
60	0.3	3593.07	**6.93**	6.91	**6.93**	6.21	0.00	5.22	0.13
	0.4	3590.68	**9.32**	9.29	9.28	7.90	0.00	1.87	0.49
	0.5	3585.29	**14.65**	14.36	14.29	12.24	0.00	0.54	2.21
	0.6	3572.61	**24.06**	23.21	23.24	20.16	0.00	1.09	12.91
	0.7	3534.62	**37.50**	36.85	35.96	31.89	0.09	5.83	26.43
	0.8	3456.59	21.90	**22.00**	21.78	21.46	21.99	7.55	19.85
70	0.3	4890.20	**9.80**	9.78	9.78	7.12	0.00	3.55	0.05
	0.4	4887.73	**12.25**	12.23	**12.25**	8.67	0.00	0.63	0.25
	0.5	4881.09	**18.48**	18.32	18.35	12.88	0.00	0.08	1.81
	0.6	4868.21	**27.98**	27.09	26.72	20.31	0.00	0.53	9.56
	0.7	4829.65	**43.30**	42.76	41.32	34.73	0.00	2.29	30.06
	0.8	4731.35	34.56	**35.32**	34.46	32.58	30.09	6.38	29.82

Proof: Let $u \in S$ be any 2-tight vertex, x, y be u'-s solution neighbors, and F denote the set of vertices free from $(S \setminus \{x\}) \cup \{u, v\}$. Similarly to Theorem 3, it suffices to show that an MIS of F can be computed in $O(n)$ time.

Suppose that x and y are u'-s solution neighbors in the directions d_x and d_y respectively. Here we give the proof only to the case $(d_x, d_y) = (1, 2)$. The other cases are analogous.

We construct a bipartite graph that consists of four vertices corresponding to solid grid lines $\ell_{x,1}$, $\ell_{y,2}$, $\ell_{x,4}$ and $\ell_{y,4}$ and n vertices corresponding to colors in a given palette. Let F' denote the subset of vertices in F that are among $\ell_{x,3}$ and $\ell_{y,3}$. We define the edge set as follows.

- If there is 1-tight vertex $v \in F \setminus F'$ that is on $\ell_{x,1}$, then we join $\ell_{x,1}$ and $\mathrm{col}(v_1, v_2)$ by an edge.
- If there is 1-tight vertex $v \in F \setminus F'$ that is on $\ell_{y,2}$, then we join $\ell_{y,2}$ and $\mathrm{col}(v_1, v_2)$ by an edge.
- If there is 1-tight vertex $v \in F \setminus F'$ that is neither on $\ell_{x,1}$ nor on $\ell_{y,2}$ but has the same color as x, then we join $\ell_{x,4}$ and $\mathrm{col}(v_1, v_2)$ by an edge.
- If there is 1-tight vertex $v \in F \setminus F'$ that is neither on $\ell_{x,1}$ nor on $\ell_{y,2}$ but has the same color as y, then we join $\ell_{y,4}$ and $\mathrm{col}(v_1, v_2)$ by an edge.

See Fig. 5 for example. Similarly to Theorem 3 a matching in this bipartite graph corresponds to an independent set of F. We can compute a maximum matching of this graph in $O(n)$ time. It is an MIS candidate of F, but F may have a 2-tight vertex; it is w in the example of Fig. 5. This should belong to every MIS of F when no maximum matching in the bipartite graph touches $\ell_{x,1}$ or $\ell_{y,2}$. Whether this is the case or not can be decided in $O(n)$ time. If so, we add the 2-tight vertex to the independent set. Finally, from F', we can add any 1-tight vertex on $\ell_{x,3}$ and one on $\ell_{y,3}$ to the independent set if one exists. In this way, we can decide an MIS of F in $O(n)$ time. □

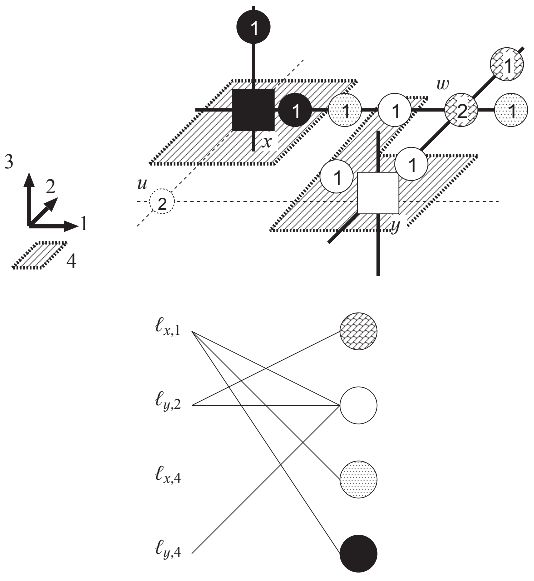

Fig. 5 An illustrative example for the proof of Theorem 4: (upper) distribution of vertices in F and (lower) the bipartite graph for this situation

5. Concluding Remarks

In this paper we reviewed the efficient local search for the PLSE problem that we proposed recently [15]. We then extended it to a variant problem, PLSE-CG, a generalization of Sudoku completion and finding OLSs.

The future work includes studying the case of $p = 3$. It is not promising to extend Trellis-swap to PLSE-CG since we may need to solve the *maximum rainbow matching problem* (a.k.a., *multiple choice matching*), an NP-hard problem [9], to determine the MIS of the trellis. Now we are working on efficient local search algorithm for the partial symmetric Latin square extension problem, which has strong application in scheduling.

References

[1] Alidaee, B., Kochenberger, G. and Wang, H.: Simple and fast surrogate constraint heuristics for the maximum independent set problem, *J. Heuristics*, Vol. 14, pp. 571–585 (2008).

[2] Andrade, D., Resende, M. and Werneck, R.: Fast local search for the maximum independent set problem, *J. Heuristics*, Vol. 18, pp. 525–547 (2012). The preliminary version appeared in Proc. 7th WEA (LNCS vol. 5038), pp. 220–234 (2008).

[3] Barry, R. A. and Humblet, P. A.: Latin routers, design and implementation, *IEEE/OSA J. Lightwave Technology*, Vol. 11, No. 5, pp. 891–899 (1993).

[4] Barták, R.: On generators of random quasigroup problems, *Proc. CSCLP 2005*, pp. 164–178 (2006).

[5] Colbourn, C. J.: The complexity of completing partial Latin squares, *Discrete Applied Mathematics*, Vol. 8, pp. 25–30 (1984).

[6] Colbourn, C. J. and Dinitz, J. H.: *Handbook of Combinatorial De-*

signs, Chapman & Hall/CRC, 2nd edition (2006).

[7] Cygan, M.: Improved Approximation for 3-Dimensional Matching via Bounded Pathwidth Local Search, *Proc. FOCS 2013*, pp. 509–518 (2013).

[8] Fürer, M. and Yu, H.: Approximating the *k*-Set Packing Problem by Local Improvements, *Proc. ISCO 2014*, LNCS, Vol. 8596, pp. 408–420 (2014).

[9] Garey, M. R. and Johnson, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Company (1979).

[10] Glover, F. and Kochenberger, G.(eds.): *Handbook of Metaheuristics*, Kluwer Academic Publishers (2003).

[11] Gomes, C. P., Regis, R. G. and Shmoys, D. B.: An improved approximation algorithm for the partial Latin square extension problem, *Operations Research Letters*, Vol. 32, No. 5, pp. 479–484 (2004).

[12] Gomes, C. P. and Shmoys, D. B.: Completing quasigroups or Latin squares: a structured graph coloring problem, *Proc. Computational Symposium on Graph Coloring and Generalizations* (2002).

[13] Gonzalez, T. F.(ed.): *Handbook of Approximation Algorithms and Metaheuristics*, Chapman & Hall/CRC (2007).

[14] Hajirasouliha, I., Jowhari, H., Kumar, R. and Sundaram, R.: On completing Latin squares, *Proc. STACS 2007*, LNCS, Vol. 4393, pp. 524–535 (2007).

[15] Haraguchi, K.: An Efficient Local Search for Partial Latin Square Extension Problem, *Proc. CPAIOR 2015*, LNCS, Vol. 9075, pp. 182–198 (2015).

[16] Haraguchi, K. and Ono, H.: Approximability of Latin Square Completion-Type Puzzles, *Proc. FUN 2014*, LNCS, Vol. 8496, pp. 218–229 (2014).

[17] Hopcroft, J. E. and Karp, R. M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, *SIAM J. Computing*, Vol. 2, No. 4, pp. 225–231 (1973).

[18] Hurkens, C. A. J. and Schrijver, A.: On the size of systems of sets every *t* of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems, *SIAM J. Discrete Mathematics*, Vol. 2, No. 1, pp. 68–72 (1989).

[19] IBM ILOG CPLEX: `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`. accessed on January 20, 2015.

[20] Itoyanagi, J., Hashimoto, H. and Yagiura, M.: A local search algorithm with large neighborhoods for the maximum weighted independent set problem, *Proc. MIC 2011*, pp. 191–200 (2011). The full paper is written in Japanese as a master thesis of the 1st author in Graduate School of Information Science, Nagoya University (2011).

[21] Kumar, R., Russel, A. and Sundaram, R.: Approximating Latin square extensions, *Algorithmica*, Vol. 24, No. 2, pp. 128–138 (1999).

[22] LocalSolver: `http://www.localsolver.com/`. accessed on January 20, 2015.