# A Genetic Algorithm for the Picture Maze Generation Problem

Yuichi Nagata[*] Akinori Imamiya[‡] Norihiko Ono[†]

[*†] Graduate School of Technology, Industrial and Social Sciences, Tokushima University, 2-1 Minami-jousanjima, Tokushima-shi, Tokushima 770-8506, Japan
[‡] Graduate School of Advanced Technology and Science, Tokushima University, 2-1 Minami-jousanjima, Tokushima-shi, Tokushima 770-8506, Japan

## Abstract

A picture maze is a maze puzzle that reveals a hidden picture when the solution path is filled. The picture maze generation problem (PMGP) consists in generating a picture maze whose solution path draws the shape most similar to a given raster image. The PMGP can be formulated as the longest path problem (LPP) on grid graphs, and we propose a genetic algorithm (GA) for this problem. In our formulation, we optimize the start and exit positions simultaneously as well as the solution path. To construct an effective GA, we employ edge assembly crossover (EAX), which is known as a very effective crossover operator for the traveling salesman problem (TSP). However, because of the difference between the two problems, we adapt EAX to the PMGP in a novel manner. The proposed GA can generate satisfactory picture mazes in 17 s for complicated raster images with sizes up to $55 \times 105$.
*Keyword*: picture maze, genetic algorithm, edge assembly crossover, longest path problem, grid graph

## 1 Introduction

The picture maze considered in this paper is a maze puzzle that reveals a hidden picture when the solution path is filled. An example is shown in Fig. 1. The original concept of picture mazes of this type was devised in Japan, and many manually created picture mazes are printed in Japanese puzzle magazines [1, 2]. Conceptis Ltd. developed an algorithm to generate picture mazes automatically and published several books that include many

*Corresponding author. Tel: +81-88-656-7505, E-mail: nagata@is.tokushima-u.ac.jp

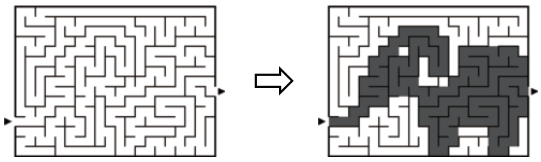

Fig. 1: Example of a picture maze. (Source: `http://wiki.logic-masters.de`.)

picture mazes generated by this algorithm, which are called Maze-a-Pix puzzles in those books [3]. This algorithm, however, has not been made public.

As an attempt to generate picture mazes automatically, Okamoto and Uehara [4] formulated the picture maze generation problem (PMGP)[1] as follows. For an input black-and-white raster image with $m$ rows and $n$ columns, construct a maze in which the image obtained by filling the solution path matches the input image. They developed a solution method for the PMGP, which was further improved by Hamada [5]. In these methods, however, each pixel of the input image must be divided into $2 \times 2$ pixels to guarantee the existence of a solution path passing through all the black pixels.

Ikeda and Hashimoto [6] formulated the PMGP in a different manner to generate picture mazes without changing the size of the input image. Instead, they accepted discrepancies between the input image and the pixels filled by the solution path, where the objective was to find a maze including a solution path with as few discrepancies as possible. They developed a simulated annealing (SA) algorithm for this problem. Kurokawa [7] also proposed a construction algorithm for almost the same problem, which grows a solution path by successively inserting path segments.

In this paper, we propose a genetic algorithm (GA) for the PMGP formulated by Ikeda and Hashimoto. Although the basic structure of the proposed GA is simple, we develop a crossover operator suitable for the PMGP. Because of the similarity between the PMGP and the traveling salesman problem (TSP), we employ edge assembly crossover (EAX) [8, 9], which is known as a very effective crossover operator for the TSP. However, because of the difference between the two problems, we needed to adapt EAX to the PMGP in a nontrivial manner. Furthermore, we extended the PMGP to include optimizing the start and exit positions simultaneously, whereas these positions were given in advance or had to be adjacent to each other in the previous studies [4, 5, 6, 7]. This extension makes it possible to reduce the number of discrepancies between the input image and an optimal solution path, while, at the same time, this makes it difficult to optimize the problem.

The PMGP is closely related to the longest path problem (LPP) in grid graphs (subgraphs of an infinite two-dimensional grid), a problem of finding a simple path of maximum length in a given grid graph. Given that the Hamilton path problem in grid graphs, with or without specified end points, is NP-complete [10], no polynomial-time algorithm exists for the LPP in grid graphs (unless $P = NP$) because the Hamilton path problem is a special

[1] They called this problem the picturesque maze generation problem in their paper.

case of the LPP [11]. Many polynomial-time exact algorithms [12, 13, 20, 14, 15, 16, 17] and approximating algorithms [18, 19, 21, 22, 23, 24, 25] have been developed for the LPP for special classes of graphs. A non-polynomial-time exact algorithm for general graphs has also been proposed [26]. In regard to the LPP for grid graphs, Keshavarz-Kohjerdi et al. [15] proposed a linear-time exact algorithm that finds the longest path between any two given vertices in rectangular grid graphs, and this was further improved by Keshavarz-Kohjerdi and Bagheri [17]. Zhang and Liu [24] developed a quadratic-time approximation algorithm that finds a path of length at least $\frac{5}{6}n+2$ (where $n$ is the number of the vertices) in grid graphs that possess a certain property. A relatively small number of literature studies are available on the development of approximate (heuristic) algorithms for the LPP, even if it is not limited to grid graphs, where GA [27], tabu search algorithm [26], and greedy best-first search algorithms [28] were developed. As we demonstrate using our experimental results, the GA proposed in this paper can also be used as an efficient approximate (heuristic) algorithm for the LPP in grid graphs.

The remainder of this paper is organized as follows. In Section 2, the preliminaries and background are presented, followed by the related works. We explain the crossover operator and the GA framework in Section 3. Experimental results are presented in Section 4. Finally, conclusions are given in Section 5.

## 2 Related works

### 2.1 Preliminaries and background

First, we describe the representation of the maze, where we consider a maze drawn in a rectangle with $m$ rows and $n$ columns. We regard each cell in the rectangle as a vertex and define an $m \times n$ rectangle grid graph $G$ (Fig. 2(a)). Any maze can be represented as a spanning tree on $G$ (Fig. 2(b)), where an edge of the spanning tree corresponds to the passage between the adjacent cells (Fig. 2(c)). Two vertices, $s$ and $e$, which respectively represent the start and exit of the maze, can be selected arbitrarily from the vertices in the cells adjacent to the outer wall. A path from the start to the exit is uniquely determined on the spanning tree, and this constitutes the solution path of the maze.

Okamoto and Uehara [4] formulated the PMGP as follows (see also Fig. 3). For a given rectangular black-and-white raster image (input image) with $m$ rows and $n$ columns, the corresponding rectangular grid graph $G$ is defined (Fig. 3(a)). Let the vertices of $G$ be denoted by $V = V_b \cup V_w$, where $V_b$ and $V_w$ are defined as sets of the vertices assigned to the black and white cells, respectively. Let a *perfect solution path* be defined as a solution path (of the maze) that goes through all the vertices of $V_b$ without passing through any vertex of $V_w$. If a spanning tree on $G$ including a perfect solution path is found, we can easily obtain a maze including the perfect solution path by adding incorrect paths (Fig. 3(b)). In fact, paths of the spanning tree other than the solution path (i.e., incorrect paths) can be randomly generated (this is always possible if $G$ is connected), and, therefore, the

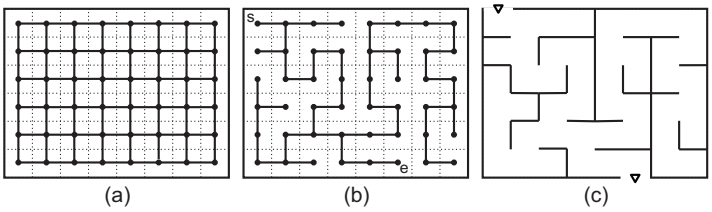


Fig. 2: Representation of the maze: (a) a rectangle with $m$ rows and $n$ columns and the corresponding grid graph $G$, (b) a spanning tree on $G$, and (c) the corresponding maze.

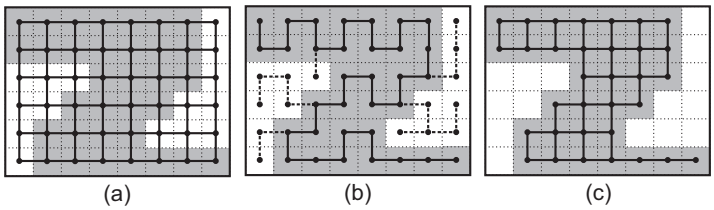


Fig. 3: Example of the PMGP: (a) an input image and the corresponding grid graph $G$, (b) a spanning tree on the grid graph $G$ including a perfect solution path, and (c) the foreground subgraph.

generation of a solution path can be considered separately from the generation of incorrect paths. A subgraph on $G$ induced by $V_b$ is called a *foreground subgraph* (Fig. 3(c)). As a consequence, finding a perfect solution path is equivalent to finding a Hamilton path on the foreground subgraph. For a given input image, however, a Hamilton path on the foreground subgraph may not exist or its discovery may be practically impossible, even if such a path exists, because the Hamilton path problem for grid graphs, with or without specified end points, is NP-complete [10].

### 2.2 Previous approaches

For a given rectangular black-and-white raster image (input image), Okamoto and Uehara [4] proposed a method to generate mazes including a perfect solution path by increasing the size of the input image. Fig. 4 illustrates the procedure of this method. For an input image, a random spanning tree on the foreground subgraph is constructed (Fig. 4(a)). Then, each pixel of the input image is divided into $2 \times 2$ pixels, yielding an extended raster image with $2m$ rows and $2n$ columns (Fig. 4(b)). A Hamilton path of the foreground subgraph defined on the extended image can then be found easily, as illustrated in Fig. 4(c); starting from a randomly selected vertex adjacent to the outer wall, a path on the foreground subgraph is constructed such that it always passes on one side of the spanning tree on the original foreground subgraph.

After constructing a solution path (Hamilton path) on the foreground subgraph for the

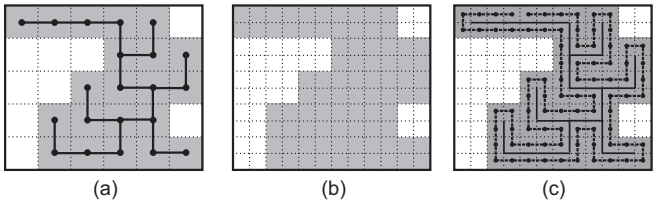


Fig. 4: Okamoto and Uehara's method: an input image and a random spanning tree on the foreground subgraph, (b) the extended input image, and (c) a Hamilton path on the foreground subgraph of the extended image (dotted line).

extended image, Okamoto and Uehara first constructed a spanning tree on the $2m \times 2n$ grid graph by adding paths randomly to the solution path. They also presented a heuristic method to merge incorrect paths into longer ones to improve the quality of the mazes because the existence of short incorrect paths renders them boring.

Because of the nature of Okamoto and Uehara's method, however, the shape of a hidden picture is inevitably jagged and the maze size becomes unnecessarily large because each pixel of the original image is divided into $2 \times 2$ pixels to construct the extended image. Moreover, the start and exit positions must be adjacent to each other and the created mazes are always solved without meeting a dead end simply by following the path according to the right-hand (or left-hand) rule. Hamada [5] improved Okamoto and Uehara's method and reduced the latter drawback, which allows the start and exit to be located at any positions given in advance if the foreground subgraph has a certain property (two-edge-connected).

As stated by Zhang and Liu [24], the PMGP can be formulated as the LPP in a foreground subgraph, i.e., to find a simple path of maximum length on the foreground subgraph. In this case, we do not need to increase the size of the input image (and the maze size); however, we have to allow discrepancies between the black cells of the input image and the solution path. They proposed a quadratic-time approximation algorithm that finds a path of length at least $\frac{5}{6}n + 2$ (where $n$ is the number of vertices) in grid graphs that possess a certain property (a square-free two-factor) and generated picture mazes using this algorithm. Because of the nature of this method, however, the start and exit positions must be adjacent to each other.

Ikeda and Hashimoto [6] formulated the PMGP as a different optimization problem. In their formulation, a solution path can go through vertices in both black and white cells. The objective is to find a solution path starting and ending at given positions that minimizes the discrepancy between the input image and the solution path. Let $P$ be a solution path of the maze and $I_P(v),\ v \in V$, be a function that returns 1 if $P$ visits $v$ or 0 otherwise. The degree of discrepancies is then defined by

$$eval_1(P) = \sum_{v \in V_b} w_v(1 - I_P(v)) + \sum_{v \in V_w} w_v I_P(v), \tag{1}$$

where $w_v$ is a weight assigned to vertex $v$. In a study reported in [6], $w_v$ was set to 1.0 if $v$ was in a black cell, 2.0 if $v$ was in a white cell adjacent to at least one black cell, or 100.0 otherwise. This weight setting was designed to prevent the solution path from protruding greatly from the area of the black cells. Ikeda and Hashimoto [6] developed an SA algorithm for solving the formulated problem. We refer to this variant of the PMGP simply as the PMGP in this paper and propose a GA for this problem.

# 3 Proposed method

We propose a GA for the PMGP formulated by Ikeda and Hashimoto [6] (see Section 2.2). In their formulation, the start and exit positions are given in advance. However, the minimum possible objective value depends on these positions and we therefore attempt to optimize them and the solution path simultaneously.

In this section, we first propose a crossover operator suitable for this problem and describe the GA framework. In addition, we describe nice properties of the proposed crossover operator, which are useful for efficiently generating promising offspring solutions.

## 3.1 Edge assembly crossover for the PMGP

The most important ingredient of the proposed GA is a crossover operator. For the PMGP, it would be reasonable to design a crossover operator that combines segments of two solution paths selected as parents to generate offspring solutions (new solution paths). To design such a crossover operator, we employed EAX [8, 9], which was originally designed for the TSP, because the PMGP has certain similarities to the TSP and EAX is known as a very effective crossover operator for the TSP.

To give the flavor of EAX, Fig. 5 illustrates an outline of EAX for the TSP. From a selected pair of parent solutions (tours), EAX can generate a number of offspring solutions (tours) through two phases. In the first phase, intermediate solutions are constructed by combining the edges of the parent solutions under the relaxed constraint that exactly two edges are linked to every vertex. Therefore, an intermediate solution typically consists of several sub-tours. In the second phase, each intermediate solution is modified into a Hamilton cycle by merging the sub-tours heuristically.

There are two major differences, described below, between the TSP and the PMGP, and EAX cannot be applied to the PMGP in its original form.

1. A solution candidate is a cycle in the TSP, whereas it is a path in the PMGP and the end points may not be given in advance.
2. Solution candidates pass through a set of the same vertices (all vertices) in the TSP, whereas they may pass through different sets of vertices in the PMGP.

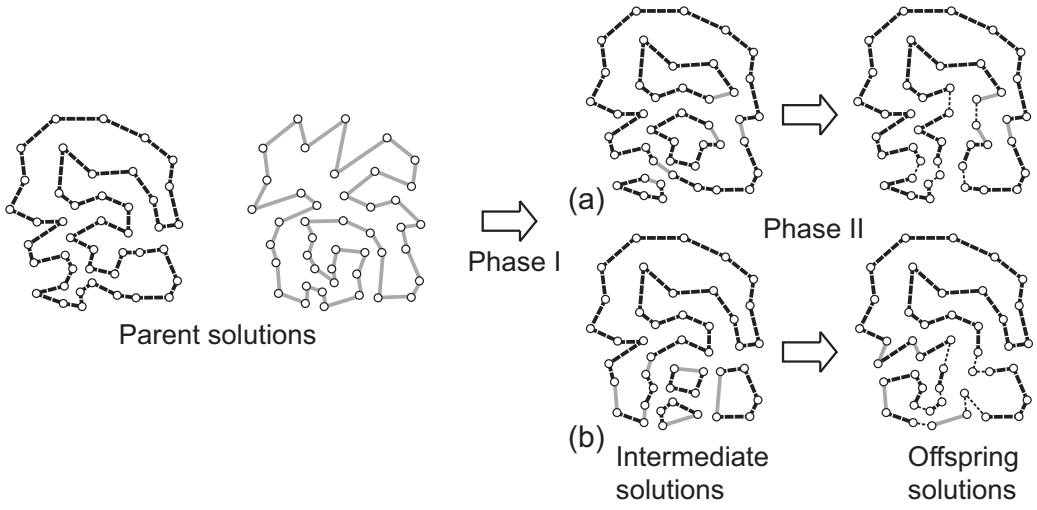


Fig. 5: Outline of EAX for the TSP.

We first describe how EAX is adapted to the PMGP (see also Fig. 6 for illustration) and define some notations required to describe EAX for the PMGP. Let the PMGP be defined on a rectangular grid graph $G = (V, E)$, where $V$ is a set of the vertices and $E$ is a set of the edges. We denote two solution paths selected as parents for EAX by $p_A$ and $p_B$. We define a graph $G_{AB} = (V, E_A \cup E_B)$, where $E_A$ ($E_B$) is defined as the union of a set of the edges that constitute $p_A$ ($p_B$) and an edge temporarily added to connect both end points of $p_A$ ($p_B$). Let $V'$ be defined as a set of the vertices of $V$ that are linked by exactly two edges of $E_A$ and exactly two edges of $E_B$ on $G_{AB}$. We consider representing a path segment consisting of edges of $E_A$ ($E_B$) that have both end points in $V'$ (and do not pass through any vertex of $V'$) as an edge. Then, we define a graph $G'_{AB} = (V', E'_A \cup E'_B)$, where $E'_A$ ($E'_B$) is defined as a set of the edges obtained from $E_A$ ($E_B$) by this procedure (see Fig. 6). Note that an element of $E'_A$ ($E'_B$) is defined as an edge of the graph $G'_{AB}$, and, therefore, it should be represented as a line segment between the end points. However, it is illustrated as a segment consisting of edges of $E_A$ (or $E_B$) in the example presented in Fig. 6 for an intuitive understanding of the EAX procedure described later. From the definition of $G'_{AB}$, any two solution paths $p_A$ and $p_B$ can always be converted into two Hamilton cycles on $G'_{AB}$. Therefore, we can apply EAX for the TSP to the converted Hamilton cycles, where the edges of $E'_A$ and $E'_B$ are combined to generate offspring solutions.

Next, we describe the detailed procedure of EAX adapted to the PMGP (see also Fig. 6). EAX generates offspring solutions (solution paths) through the following five steps, where Steps 2–4 are the same as in the EAX procedure for the TSP.

**EAX procedure:**

**1.** Let $G = (V, E)$ be the rectangular grid graph that corresponds to the given input image. For a given solution paths $p_A$ and $p_B$, construct $G_{AB} = (V, E_A \cup E_B)$ and

convert it into $G'_{AB} = (V', E'_A \cup E'_B)$.

**2.** Partition all edges of $G'_{AB}$ into *AB-circuit*s, where an AB-circuit[2] is defined as a circuit obtained by tracing the edges of $E'_A$ and $E'_B$ alternately on $G'_{AB}$. This partition is always possible, but it is not uniquely determined. The edges are therefore randomly partitioned into AB-circuits.

**3.** Construct a number of *E-set*s by selecting AB-circuits according to a given selection strategy, where an E-set is defined as an arbitrary combination of AB-circuits.

**4.** For each E-set, generate an intermediate solution from $p_A$ by removing the edges of $E'_A \cap$ E-set and then adding the edges of $E'_B \cap$ E-set. Because for every vertex on $G'_{AB}$ the number of edges added is equal to the number of edges removed, exactly two edges are incident to every vertex in an intermediate solution. Therefore, an intermediate solution consists of a Hamilton cycle or more than one sub-tour on $G'_{AB}$.

**5.** Each intermediate solution is modified into a path on $G$ through the following procedure.

**5-1.** We consider the intermediate solution on $G_{AB}$. That is, each edge of the intermediate solution (consisting of edges of $E'_A$ and $E'_B$) is converted into the corresponding edges of $E_A$ and $E_B$. Then, the temporal edge is removed (if it exists). At this point, the intermediate solution consists of a path and possibly one or more sub-tours on $G$.

**5-2.** If the intermediate solution is a path on $G$ (i.e., there is no sub-tour), terminate Step 5 (offspring solution is obtained). Otherwise, select a sub-tour randomly.

**5-3.** The selected sub-tour is merged into the path or one of the other sub-tours (if it exists) by removing one edge from the selected sub-tour and one edge from the path or other sub-tours followed by adding two edges of $E$ to connect the breakpoints. Here, the two edges to be removed must be parallel and next to each other on $G$ (so that the breakpoint can be connected with two edges of $E$). If there is more than one candidate that can merge the selected sub-tour, the join location is selected randomly from among the possible candidates because the objective value of the offspring solution does not depend on the join location.

**5-4.** Return to Step 5.2.

In Step 2, all the edges of $G'_{AB}$ are partitioned into AB-circuits, some of which may consist of multiple edges consisting of two edges, one from $E'_A$ and one from $E'_B$. For example, there are eight such AB-circuits in the example presented in Fig. 6. Such an AB-circuit is called an *ineffective* AB-circuit because it does not affect the intermediate solution if it is included in an E-set. All ineffective AB-circuits are therefore discarded.

[2]In the original papers [8, 9], an AB-circuit was called an AB-cycle. However, it should be called an AB-circuit according to graph theory.

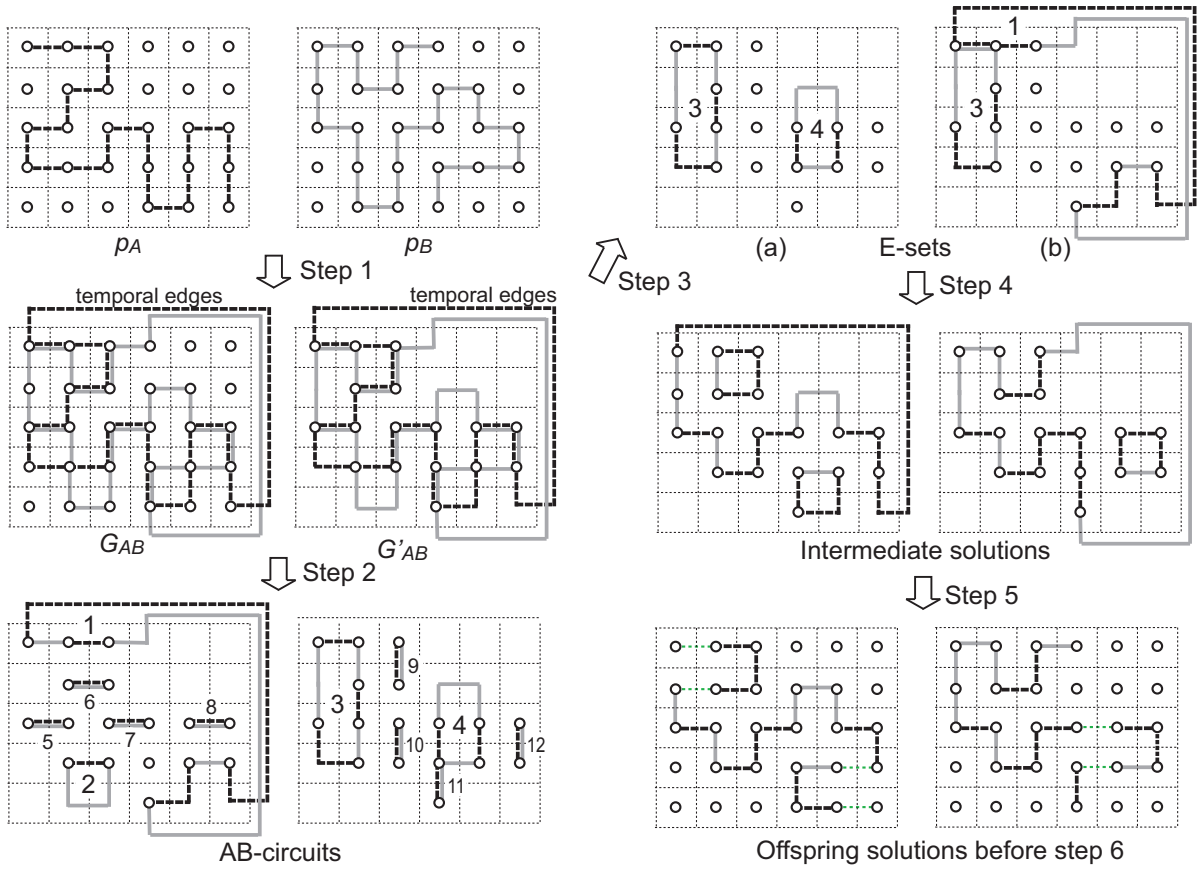


Fig. 6: Edge assembly crossover for the PMGP.

In fact, Step 5-3 does not always succeed because there may be no possible join location to connect the selected sub-tour. Fig. 7 shows typical examples of such a failure (these examples do not correspond to $p_A$ and $p_B$ shown in Fig. 6). In this case, such an intermediate solution is discarded and an infeasible solution is returned. In our observations, such failures did not occur frequently and did not become a fatal problem.

If an E-set includes no temporal edge (e.g., E-set (a) in Fig. 6), the resulting intermediate solution necessarily includes the temporal edge added to $p_A$, but does not include the temporal edge added to $p_B$. On the other hand, if an E-set includes both temporal edges (e.g., E-set (b)), the resulting intermediate solution necessarily includes the temporal edge added to $p_B$, but does not include the temporal edge added to $p_A$. Therefore, an offspring solution is a path with end points inherited from either $p_A$ or $p_B$.

Fig. 6 shows an example where both temporal edges are included in the same AB-circuit. In fact, two temporal edges may belong to different AB-circuits, as illustrated in Fig. 8, which can in fact be generated from the parent solutions shown in Fig. 6. In this case (this situation rarely occurs), we need to pay attention to the selection of AB-

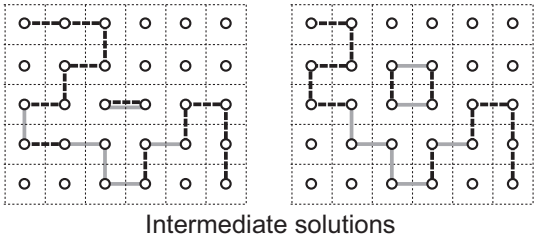


Fig. 7: Examples of intermediate solutions where Step 5-3 fails.

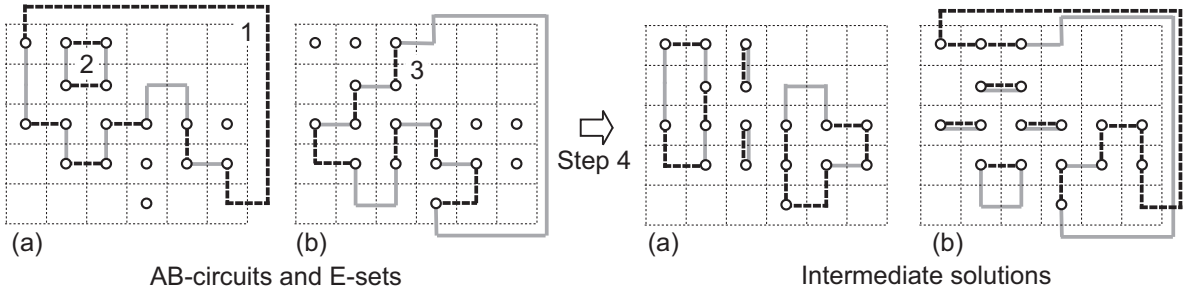


Fig. 8: (Left) Example of AB-circuits generated from the parent solutions $p_A$ and $p_B$ given in Fig. 6. This also illustrates two E-sets. (Right) Intermediate solutions obtained by applying the two E-sets.

circuits for constructing E-sets. If an E-set includes only the temporal edge added to $p_A$, the resulting intermediate solution includes no temporal edge (Fig. 8(a)). Conversely, if an E-set includes only the temporal edge added to $p_B$, the resulting intermediate solution includes both temporal edges (Fig. 8(b)). In both cases, the intermediate solution cannot be modified into a path. To avoid these situations, if the two temporal edges are included in different AB-circuits, we combine the two AB-circuits and regard them as one.

EAX can generate various intermediate solutions, depending on the combination of AB-circuits selected in Step 3 for constructing E-sets, and several strategies for the selection of AB-circuits have been proposed for the TSP [9]. For the PMGP, we employ the "single" strategy, which selects one AB-circuit randomly to construct an E-set (without overlapping the previous selection if more than one offspring solution is generated). It is also possible to select more than one AB-circuit to construct an E-set, which is in fact useful for the TSP [9]. However, we use only the single strategy because there is no advantage in selecting more than one AB-circuit for the PMGP (details are given in Section 3.4). For every AB-circuit generated in Step 2 of the EAX procedure, we construct an E-set in the proposed GA and generate an offspring solution.

### 3.2 Framework of the genetic algorithm

The framework of the proposed GA is based on the GA [9] in which the original EAX was incorporated because this framework is suitable for the use of EAX. As in the original framework, we use a population diversity management strategy, but we introduce a new strategy that is more suitable for the PMGP.

Algorithm 1 shows the framework of the proposed GA. The population consists of $N_{pop}$ solutions. The initial population members are generated by an appropriate procedure (line 1). At each generation of the GA (lines 3–9), each of the population members is selected once as parent $p_A$ and once as parent $p_B$, in random order (lines 3 and 5). Note that for $i = N_{pop}$, $p_B(= x_{r(1)})$ is already updated in the same generation. This is just to simplify the implementation. For each pair of parents, $N_{ch}$ offspring solutions are generated using EAX (line 6), where $N_{ch}$ is the number of AB-circuits generated in Step 2 of the EAX procedure. Then, we select the best individual among the generated offspring solutions and $p_A$ according to a given evaluation function and replace $x_{r(i)}$ $(= p_A)$ with the selected individual (line 7). Then, the end points of $x_{r(i)}$ are locally improved (details are described later) when the start and exit positions are optimized simultaneously (line 8). The main loop (lines 3–9) is repeated until a termination condition is met (line 10). Finally, the best individual in the population is returned (line 11).

**Algorithm 1** : Procedure of the genetic algorithm

```
 1: Generate initial population x_1, ..., x_{N_pop};
 2: repeat
 3:    r(·) := a random permutation of 1, ..., N_pop;
 4:    for i := 1 to N_pop do
 5:       p_A := x_{r(i)}, p_B := x_{r(i+1)}; // r(N_pop + 1) = r(1)
 6:       {c_1, ..., c_{N_ch}} := EAX(p_A, p_B);
 7:       x_{r(i)} := SELECT_BEST(c_1, ..., c_{N_ch}, p_A);
 8:       x_{r(i)} := IMPROVE_ENDPOINTS(x_{r(i)}); // used only for the free setting
 9:    end for
10: until the termination condition is satisfied
11: return the best individual in the population;
```

We apply the GA to the PMGP under either of the following two conditions.

- **Fixed setting**: The start and exit positions are given in advance and fixed during the search.
- **Free setting**: The start and exit positions are also optimized simultaneously.

We need only to change the procedure for generating the initial population members depending on these conditions. For the fixed setting, all initial population members must

have the same end points, which are specified in advance. Because of the nature of EAX for the PMGP, all offspring solutions have the same end points as the specified end points. For the free setting, on the other hand, we generate initial population members having a variety of end points. Owing to the nature of EAX, each offspring solution has end points of either $p_A$ or $p_B$, and the number of individuals (solution paths) having promising end points will increase in the population as the generation of the GA proceeds. We describe the method for generating the initial population in the following subsection.

However, if only EAX is used to generate new individuals, an expected drawback of the proposed GA for the free setting is that possible combinations of the start and exit positions can only be introduced during the generation of the initial population. Therefore, for each individual $x_{r(i)}$ updated by an offspring solution (line 7), we try to improve its end points (line 8), which makes it possible to introduce new start and exit positions in the population. This procedure simply checks the end points of the input individual (solution path) and makes a local move of each end point only when the evaluation value can be improved. Possible local moves are defined by neighborhood (d) illustrated in Fig. 9, where each end point can be shifted along the outer wall (details are described in Section 3.3).

Let $y$ denote an individual among the offspring solutions and $p_A$ evaluated for selecting the best individual to replace $x_{r(i)}$ (line 7). The most straightforward evaluation function is the objective function $eval_1(y)$ (Eq. (1)), and we use it as the first candidate. In [9], an elaborate evaluation function was employed, aimed at maintaining population diversity in a positive manner. In this paper, we propose a different evaluation function suitable for the PMGP for the same purpose. To explain this evaluation function, let us denote the population by *pop* and define a function $E(pop)$, which evaluates the desirability of the population. Let $f(v),\ v \in V$, be the frequency of the population members that pass through vertex $v$. Then, $E(pop)$ is defined as

$$E(pop) = \sum_{v \in V_b} w_v (N_{pop} - f(v))^2 + \sum_{v \in V_w} w_v f(v)^2. \tag{2}$$

We evolve the population so that this function value is minimized. The aim behind this is basically to increase (resp. decrease) the number of population members that pass through vertex $v \in V_b$ (resp. $V_w$). In addition, this function is sensitive to the change in $f(v)$ having a relatively small (resp. large) value for $v \in V_b$ (resp. $V_w$). This property is intended to avoid exterminating (resp. extremely increasing) the individuals passing through $v \in V_b$ (resp. $V_w$). For example, when the value of $f(v)$ becomes zero (resp. $N_{pop}$), the opportunity to generate individuals passing (resp. not passing) through $v$ by EAX is completely lost. Let $pop(y)$ denote the population obtained from *pop* by replacing the population member $x_{r(i)}$ (selected as $p_A$) with an individual $y$. Then, we evaluate each individual $y$ according to the following evaluation function:

$$eval_2(y) = E(pop(y)) - E(pop). \tag{3}$$

In terms of the evaluation function and termination condition, we compare the following two strategies.

- **Greedy selection**: Evaluation function $eval_1(y)$ is used. The GA is terminated when the best solution in the population is not improved over 100 successive generations.

- **Diversity-preserving selection**: Evaluation function $eval_2(y)$ is used from the beginning of the GA until no improvement of the best solution is found over 50 successive generations. Then, evaluation function $eval_1(y)$ is used until no improvement in the best solution is found over 50 successive generations. Then, the GA is terminated.

The values of the parameters for the termination conditions were set to sufficiently large values rather than to tuned ones because there is no particular disadvantage (except for the computation time) if these values are larger than necessary.

We tested the following three weight settings for the two evaluation functions:

- **Weight setting 1**: $w_v = 1$ $(v \in V)$.

- **Weight setting 2**: $w_v = 2$ if $v$ is in a black cell not adjacent to the outer wall and its four neighboring cells are also black; otherwise, $w_v = 1$. This setting is used to eliminate discrepancies in the internal area of the black cells because such discrepancies are more noticeable than those in the boundary area.

- **Weight setting 3**: $w_v = 1$ $(v \in V_b)$ and $w_v = 4$ $(v \in V_w)$. We use this setting when the GA is used as an approximate solution method for the LPP. More details are provided in the experimental results section.

After a solution path is obtained by the GA, the remaining paths (incorrect paths) are first generated randomly and then merged into longer ones to improve the quality of the maze in the same manner as that described in [4] (see Section 2.2). We then generate the corresponding maze.

### 3.3 Generation of the initial population

To generate the initial population of the GA, we use a simplified version of the SA algorithm developed by Ikeda and Hashimoto [6] for the PMGP. We first describe the procedure of the original SA algorithm.

**SA algorithm:**

**1.** Make an initial solution $P$.

**2.** Initialize the temperature $T := T_{init}$.

**3.** Select a neighbor solution $P'$ randomly from neighborhood $\mathcal{N}(P)$.

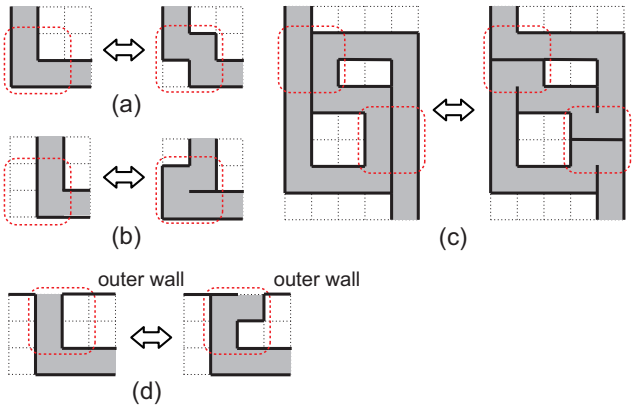


Fig. 9: Neighborhood structures used in the SA algorithms. Moves in both directions are possible.

**4.** If $eval_1(P') - eval_1(P) \leq 0$, replace $P$ with $P'$; otherwise, replace $P$ with $P'$ with a probability of $\exp\{-\frac{eval_1(P')-eval_1(P)}{T}\}$.

**5.** Set $T := \gamma T$, where $\gamma$ is a cooling factor. If $T_{end} < T$, return to Step 3; otherwise, terminate the procedure and output the best solution found so far.

Fig. 9 illustrates three neighborhood structures used in the original SA algorithm (neighborhood (d) is not used in the original SA). Neighborhood (a) (resp. (b)) operates on a block of $2 \times 2$ cells (indicated by a dotted square) including three (resp. two or four) cells through which the solution path passes and makes the possible local move. While neighborhood (b) changes the path length, neighborhood (a) does not. Neighborhood (c) operates on two different blocks of $2 \times 2$ cells (indicated by dotted squares), each including four cells through which the solution path passes, and changes the direction of the walls in the selected blocks. This operation does not change the cells passed by the solution path, but it is useful for diversifying the search. Because of the high computational cost for neighborhood (c), however, this neighborhood was selected with a probability of 0.1 and a composite of neighborhoods (a) and (b) was selected with a probability of 0.9 to construct $\mathcal{N}(P)$.

The simplified SA algorithm used for generating the initial population of the GA is basically the same as the original SA, except that neighborhood (c) is eliminated because it is time consuming. For the fixed setting, we used the simplified SA as it is, i.e., neighborhood $\mathcal{N}(P)$ is defined as a composite of the two neighborhoods (a) and (b). For the free setting, we modified the simplified SA to optimize the positions of the end points simultaneously. For this purpose, we introduced neighborhood (d) (see Fig. 9), because the two neighborhoods (a) and (b) cannot change the positions of the end points. Neighborhood (d) operates on a block of $2 \times 2$ cells including one of the end points and shifts the position of the end point along the outer wall. Consequently, neighborhood $\mathcal{N}(P)$ is defined as a composite of the three neighborhoods (a), (b), and (d) for the free

setting.

Briefly stated, an initial solution for the original SA was obtained as a path starting from a given start vertex and ending at a given exit vertex in a randomly generated spanning tree on the foreground subgraph. In the simplified SA, we used a different initial solution generation procedure to generate diverse initial population members for the GA. For this purpose, we generated a random path that goes through as many vertices as possible on the rectangular grid graph $G$ as an initial solution for the SA. To obtain such a random path, we performed a local search algorithm obtained from the simplified SA (using neighborhoods (a) and (b)) simply by setting $T = 0$, with the objective function defined as the path length. An initial solution of the local search is generated in the same way as in the original SA, but in the case of the free setting, the end points are selected randomly from the cells adjacent to the outer wall. Each run of the local search was terminated when the number of iterations reached 100,000. This number was in fact larger than necessary, but we used it because the required computation time was negligible.

### 3.4 Efficient generation of offspring solutions

EAX has nice properties for efficiently generating offspring solutions that improve parent solutions. For any E-set constructed in Step 3 of the EAX procedure, we can calculate the value of the evaluation function ($eval_1(y)$ or $eval_2(y)$) for the resulting offspring solution $y$ using only the difference information between $p_A$ and $y$. More specifically, let $V_{re}$ (resp. $V_{add}$) be a subset of $V$ that is passed by $p_A$ (resp. $y$) but not by $y$ (resp. $p_A$). These sets are easily obtained from the E-set. For example, if an edge of $E'_A$ is included in the E-set, the vertices of $V$ through which the corresponding path segment passes on the graph $G$ become elements of $V_{re}$ (see Fig. 6). Conversely, if an edge of $E'_B$ is included in the E-set, the vertices of $V$ through which the corresponding path segment passes become elements of $V_{add}$. Considering that an intermediate solution and the corresponding offspring solution pass through the same vertices, we can efficiently calculate the value of $eval_1(y)$ as follows:

$$eval_1(y) = \sum_{v \in V_{re} \cap V_b} w_v - \sum_{v \in V_{re} \cap V_w} w_v - \sum_{v \in V_{add} \cap V_b} w_v + \sum_{v \in V_{add} \cap V_w} w_v. \tag{4}$$

Similarly, the value of $eval_2(y)$ is calculated as follows:

$$\begin{aligned} eval_2(y) = &\sum_{v \in V_{re} \cap V_b} w_v(2N_{pop} - 2f(v) + 1) + \sum_{v \in V_{re} \cap V_w} w_v(2f(v) + 1) \\ &+ \sum_{v \in V_{add} \cap V_b} w_v(-2N_{pop} + 2f(v) + 1) + \sum_{v \in V_{add} \cap V_w} w_v(-2f(v) + 1). \end{aligned} \tag{5}$$

We can easily calculate the above equations by assigning appropriate weights to the edges of the graph $G'_{AB}$. For example, when $eval_1(y)$ is used, if an edge of $G'_{AB}$ passes through two vertices 3 and 5 of $V$ (which are removed when constructing $G'_{AB}$) and $3 \in V_b$, $5 \in V_w$,

the weight assigned to this edge is given by $-w_3 + w_5$. That is, if this edge is added to (removed from) $p_A$ to generate an intermediate solution, this value is added (subtracted) when calculating the value of the evaluation function. Note that it is not guaranteed that a feasible offspring solution is always obtained for a constructed E-set because Step 5 of the EAX procedure may fail. Therefore, the calculated evaluation value is true only if a feasible offspring solution is obtained.

In practice, we calculate Eq. (4) or (5) for every AB-circuit. The value of the evaluation function for any E-set is then obtained simply by summing those of the selected AB-circuits (if more than one AB-circuit is selected to construct an E-set). Therefore, it is not necessary to select more than one AB-circuit to construct an E-set for finding an offspring solution that improves $p_A$ unless the amount of the improvement is considered. This is the reason why we use only the single strategy for constructing E-sets (see Section 3.1).

For each pair of parent solutions $p_A$ and $p_B$, we select the best individual (in terms of the selected evaluation function) among $p_A$ and all offspring solutions generated by EAX with the single strategy (lines 6 and 7 of Algorithm 1). By utilizing the aforementioned property of EAX, we can find the best individual efficiently without generating all offspring solutions. The actual procedure is as follows. For a selected pair of parents $p_A$ and $p_B$, the edges of $G'_{AB}$ are partitioned into AB-circuits (Step 2 of the EAX procedure), and we calculate the evaluation value (Eq. (4) or (5)) for every AB-circuit. After sorting the AB-circuits in ascending order of the evaluation value, we apply each AB-circuit (i.e., E-set when the single strategy is used) to $p_A$ in this order. For a selected AB-circuit, we generate an offspring solution by performing Steps 4 and 5 of the EAX procedure. If Step 5 is successful, we have already obtained the best offspring solution among all offspring solutions and therefore omit the procedure for generating the remaining offspring solutions. In addition, if the evaluation value of the selected AB-circuit becomes greater than or equal to zero, we stop the remaining procedure because there is no possibility to find an offspring solution with a better evaluation value than $p_A$.

# 4 Experimental results

We implemented the proposed GA in C++ in Ubuntu 14.04 Linux and executed the program codes on PCs, each with an Intel Core i7-4790 3.60 GHz CPU. In this section, we present our experimental results to assess the performance of the GA.

## 4.1 Experimental settings

In the experiment, the population size of the GA was set to 100. The configuration of the GA was determined by choosing one from each list given below to test several different configurations of the GA (see Section 3.2).

- Selection strategy: greedy selection or diversity-preserving selection.
- Start and exit positions: fixed setting or free setting.
- Weights of the objective function: weight setting 1, 2, or 3.

Note that, when the fixed setting was used, we did not give the start and exit positions in advance. Instead, for each trial, these positions were determined by selecting two vertices randomly from those in the black cells adjacent to the outer wall, and both ends were fixed during the search.

We denote the GA using a specified configuration such as $GA_{div}^{free}$, which means that the GA uses the diversity-preserving selection and the free setting. In the following, we specify the weight setting as necessary. $GA_{div}^{free}$ with weight setting 2 uses all options devised to improve the quality of solution paths for the PMGP. We also tested $GA_{gre}^{free}$ and $GA_{div}^{fix}$ with weight setting 2 and $GA_{div}^{free}$ with weight setting 1 to confirm the effect of each option employed in $GA_{div}^{free}$ with weight setting 2.

We tested the SA algorithm developed by Ikeda and Hashimoto [6] (see Section 3.3) for comparison. According to the literature, the parameters for the SA were set as follows: $T_{init} = 10.0$, $T_{end} = 0.1$, and $\gamma = 0.99995$. We confirmed that these parameter values were most suitable among all combinations of $T_{init} = \{2.0, 5.0, 10.0, 20.0, 40\}$ and $\gamma = \{0.99999, 0.99995, 0.9999\}$ for the five input images described below, where $T_{end}$ was set to $0.01T_{init}$. We then constructed a multi-start SA algorithm consisting of 100 independent runs of the SA algorithm because a single run of the SA was faster than that of the proposed GA. The multi-start SA was executed under the fixed and free settings, both with weight setting 2, and we denote these algorithms by $MSA^{fix}$ and $MSA^{free}$, respectively. Note that, for the free setting, neighborhood $\mathcal{N}(P)$ in the SA algorithm consists of the four neighborhoods (a), (b), (c), and (d). We used the same parameter values for the simplified SA used to generate the initial population of the GA.

We created five recognizable input raster images: "mouse" (50×30 ), "gorilla" (40×38), "cobra" (47 × 50), "panda" (72 × 67), and "kongou" (55 × 105), shown in Fig. 10. For each input image, we applied the GA and multi-start SA 100 times.

After describing the main results on the five input images, we present additional experimental results for a set of random raster images with known optimal values to verify whether the GA can reach an optimal solution or not for various instances. In addition, we evaluate the capability of the GA as an approximate (heuristic) solution method for the LPP in grid graphs.

### 4.2 Main results

We first describe the results when weight setting 2 was used. Table 1 shows the results of the GA with the three different configurations ($GA_{div}^{free}$, $GA_{gre}^{free}$, and $GA_{div}^{fix}$) and the

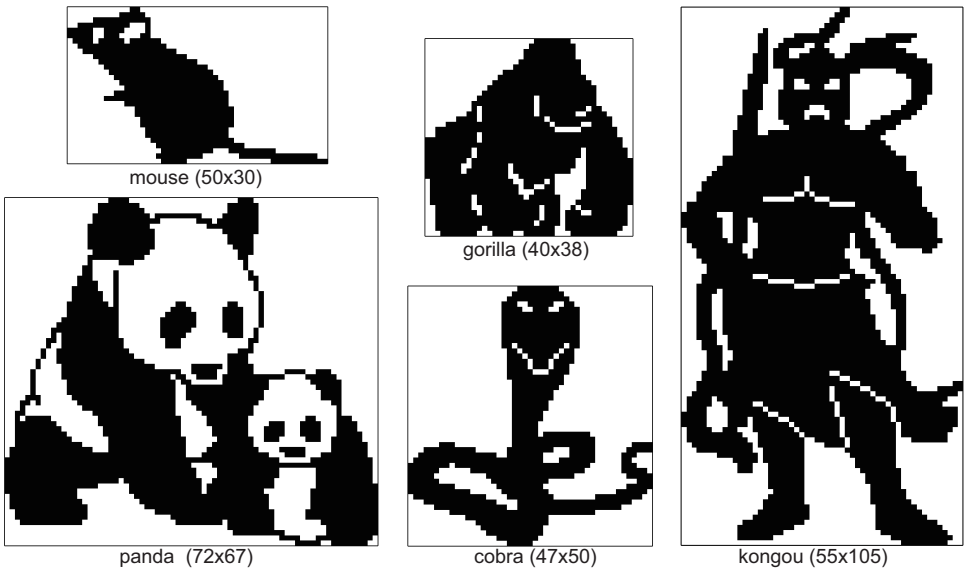


Fig. 10: Five recognizable input images

multi-start SA with the two different configurations ($MSA^{free}$ and $MSA^{fix}$) for the five input images. The table presents the best objective value (Best), the average objective value (Ave.), and the average execution time in seconds (Time). In addition, the numbers in parentheses represent the number of trials that reached the best-known objective value (found in all our experiments). The results in the table confirm that the solution quality (the values of Best and Ave.) of $GA_{div}^{free}$ is clearly better than that of $GA_{gre}^{free}$ and $GA_{div}^{fix}$. These results demonstrate the advantages of the diversity-preserving selection over the greedy selection and of the free setting over the fixed setting. However, the execution times of $GA_{div}^{free}$ were slightly greater than those of $GA_{div}^{fix}$. This was mainly due to the increase in the computational effort by the improvement procedure of the end points (line 8 of Algorithm 1) used for the free setting. When this procedure was removed from $GA_{div}^{free}$, the execution time was almost the same as that of $GA_{div}^{fix}$ and the best-known objective values were found 59, 22, 32, 21, and 57 times (not reported in the table) in the order of the five input images in Table 1.

In Fig. 11, we present the best solution paths (black lines) obtained by $GA_{div}^{free}$ with weight setting 2 for the five input images, where each cell is colored according to the following rule: light yellow (light gray in grayscale) indicates a black cell of the input image that is passed by the solution path; blue (dark gray) indicates a white cell of the input image that is passed by the solution path; and outlined black indicates a black cell of the input image that is not passed by the solution path. Therefore, discrepancies occurred in the blue (dark gray) and outlined black cells. We can see that there is no discrepancy in the internal area of the black cells of the input images, showing that weight setting 2 operates as expected.

In Table 1, we can see that the solution quality of $GA_{div}^{fix}$ (resp. $GA_{div}^{free}$) is considerably

Table 1: Results of $GA_{div}^{free}$, $GA_{gre}^{free}$, $GA_{div}^{fix}$, $MSA^{free}$, and $MSA^{fix}$ for the five input images (weight setting 2)

| | Mouse | | | Gorilla | | | Cobra | | | Panda | | | Kongou | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Best | Ave. | Time | Best | Ave. | Time | Best | Ave. | Time | Best | Ave. | Time | Best | Ave. | Time |
| $GA_{div}^{free}$ | 7 (94) | 7.1 | 3.5 | 10 (41) | 10.7 | 4.8 | 16 (32) | 16.4 | 5.1 | 48 (47) | 49.0 | 11.6 | 30 (60) | 30.4 | 16.5 |
| $GA_{gre}^{free}$ | 7 ( 7) | 8.9 | 3.4 | 12 ( 0) | 12.8 | 4.2 | 16 ( 4) | 18.4 | 4.5 | 48 ( 8) | 51.5 | 9.5 | 30 ( 2) | 34.8 | 15.0 |
| $GA_{div}^{fix}$ | 7 ( 1) | 12.7 | 3.3 | 10 ( 4) | 12.3 | 4.2 | 16 ( 3) | 19.5 | 4.3 | 49 ( 0) | 54.6 | 10.1 | 30 ( 1) | 33.2 | 13.9 |
| $MSA^{free}$ | 15 ( 0) | 22.8 | 14.9 | 27 ( 0) | 32.0 | 19.6 | 82 ( 0) | 102.5 | 22.6 | 206 ( 0) | 250.8 | 44.8 | 253 ( 0) | 280.2 | 62.9 |
| $MSA^{fix}$ | 19 ( 0) | 28.7 | 15.0 | 27 ( 0) | 32.3 | 19.9 | 65 ( 0) | 99.2 | 22.7 | 215 ( 0) | 267.1 | 45.3 | 253 ( 0) | 292.0 | 63.5 |

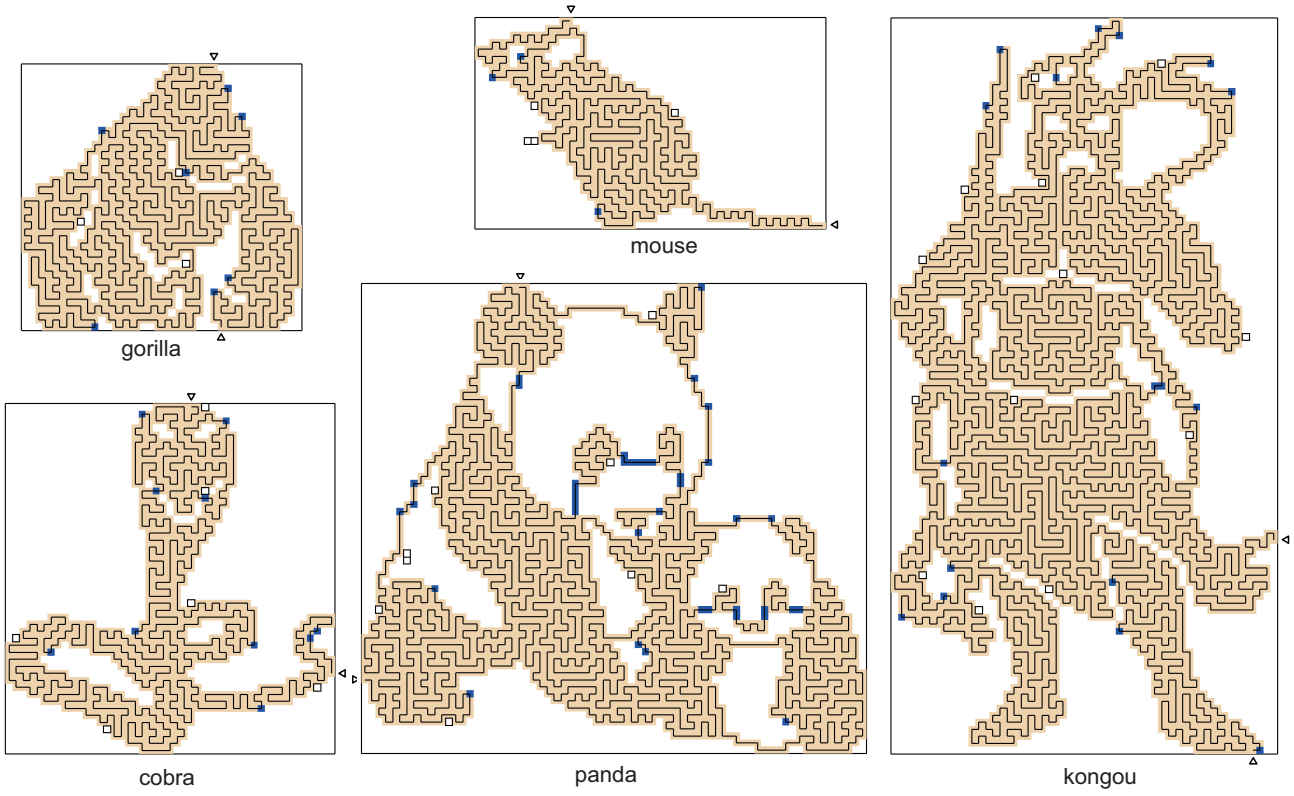


Fig. 11: Best solution paths obtained by $GA_{div}^{free}$ using weight setting 2

better than that of $MSA^{fix}$ (resp. $MSA^{free}$). Of course, it may be possible to improve the performance of the multi-start SA by using more sophisticated neighborhood structures. In our opinion, however, it is most unlikely that local search-based methods reach the best solution paths obtained by the proposed GA. This is because it is frequently necessary to change a considerable part of the current solution path to remove a discrepancy without generating a new discrepancy. On the other hand, EAX can efficiently find relatively large

Table 2: Results of $GA_{div}^{free}$ using weight settings 1 and 2

| | Mouse | | | Gorilla | | | Cobra | | | Panda | | | Kongou | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Weight | Best | Ave. | $\overline{D}_{in}$ | Best | Ave. | $\overline{D}_{in}$ | Best | Ave. | $\overline{D}_{in}$ | Best | Ave. | $\overline{D}_{in}$ | Best | Ave. | $\overline{D}_{in}$ |
| weight 1 | 7 (88) | 7.2 | 1.4 | 10 (16) | 10.9 | 4.5 | 16 (19) | 16.4 | 3.8 | 48 (45) | 49.3 | 6.4 | 30 (61) | 30.5 | 7.7 |
| weight 2 | 7 (94) | 7.1 | 0.0 | 10 (41) | 10.7 | 0.0 | 16 (32) | 16.4 | 0.0 | 48 (47) | 49.0 | 0.0 | 30 (60) | 30.4 | 0.0 |

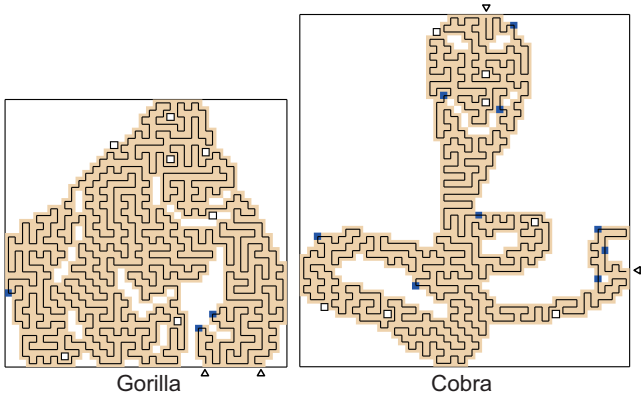


Fig. 12: Best solution paths obtained by $GA_{div}^{free}$ using weight setting 1

changes necessary to improve the parent solution $p_A$.

Next, we compare weight settings 1 and 2 to verify the effect of the latter setting on reducing the number of discrepancies in the internal area of the black cells, i.e., a set of black cells that are assigned a weight of 2. Table 2 shows the results of $GA_{div}^{free}$ using the two weight settings in the same form as in Table 1, except that the average number of discrepancies in the internal area of the black cells ($\overline{D}_{in}$) is additionally presented instead of the average execution time. The table shows that the use of weight setting 2 succeeded in completely eliminating discrepancies in the internal area of the black cells, whereas there were several such discrepancies when weight setting 1 was used. In Fig. 12, we present typical best solution paths obtained by $GA_{div}^{free}$ with weight setting 1 for two input images, gorilla and cobra; we can see that there are several discrepancies in the internal area of the black cells. Table 2 also shows that the best and average objective values of weight setting 2 are almost equal to or even better than those of weight setting 1, even though heavier weights are assigned to vertices in the internal area of the black cells in weight setting 2. This implies that the GA with weight setting 2 succeeded in eliminating discrepancies in the internal area of the black cells without increasing the total number of discrepancies achieved with weight setting 1.

Finally, we present in Fig. 13 the mazes created from the solution paths for the five input images shown in Fig. 11, where the cells passed by the solution paths are colored.

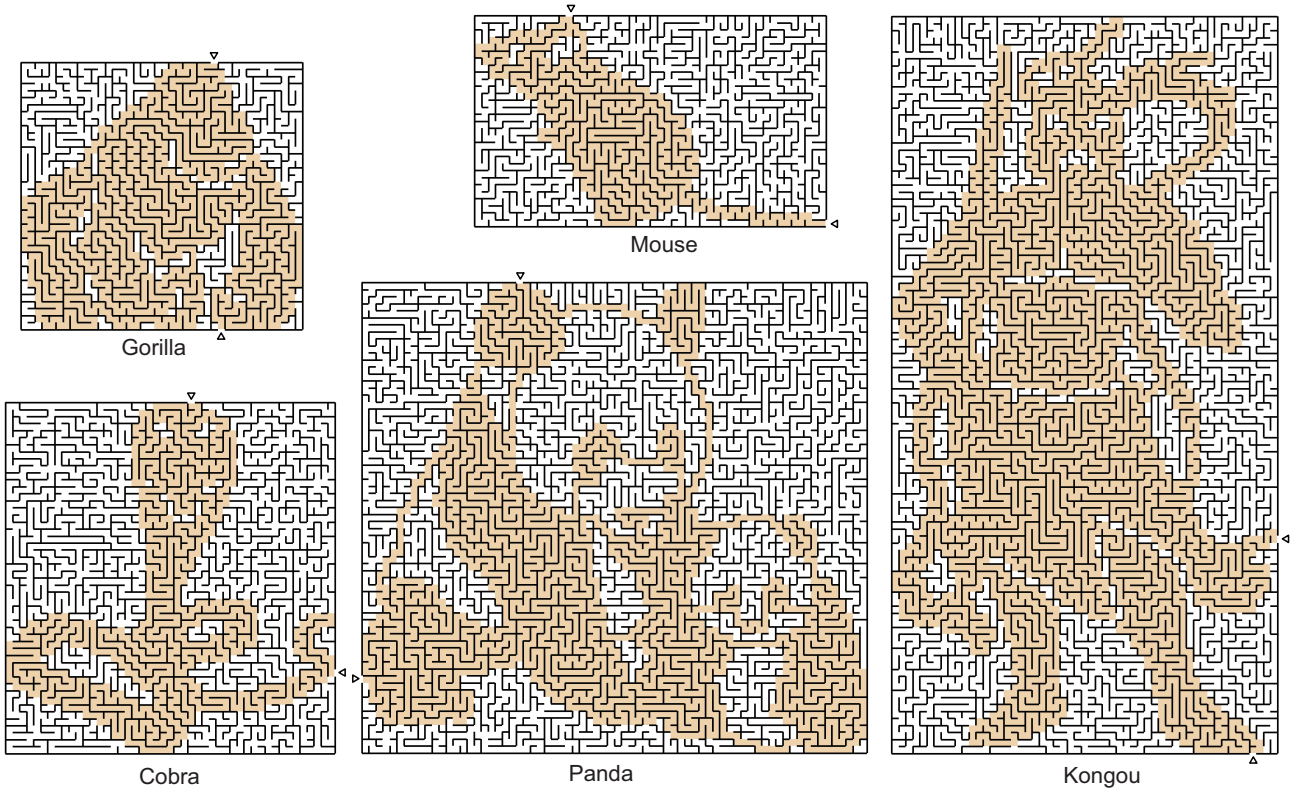


Fig. 13: Mazes created from the solution paths shown in Fig. 11

### 4.3 Results for random images

We cannot verify whether the best objective values for the five recognizable input images presented in Table 1 are optimal or not. To investigate the performance of $GA_{div}^{free}$ on input images with known optimal values, we created a set of random raster images with known optimal objective values. The basic idea is to create a random path starting and ending at two cells adjacent to the outer wall. We then obtain a raster image in which at least one perfect solution path exists by filling the cells passed by the obtained path. Briefly stated, we used the SA algorithm (the original SA algorithm described in Section 3.3) together with appropriate objective functions to generate random paths that satisfy specified properties. The actual procedure is given in Appendix A.

The random raster images created are drawn in squares and are characterized by three attributes, $size \in \{30, 50, 100\}$, $density \in \{0.5, 0.7\}$, and $Type \in \{R, C\}$. Attribute $size$ specifies the number of rows (and columns) of a raster image, and $density$ specifies the ratio of the black cells over all cells. The structure of random raster images of Type $C$ is clustered as compared with that of Type $R$. As a consequence, we had 12 different configurations for random raster images, and for each configuration we created five instances. We denote an instance with specified attribute values as ran-

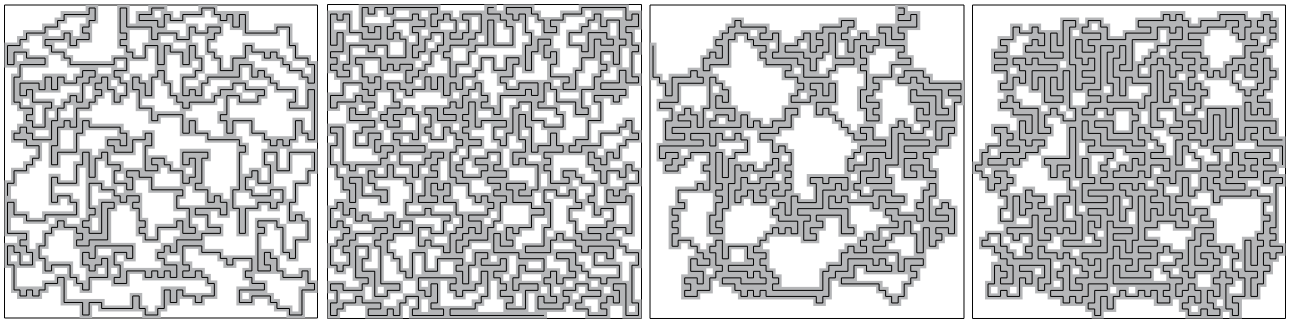


Fig. 14: Examples of random raster images and perfect solution paths

Table 3: Results of $GA_{div}^{free}$ on the random images (end points were optimized simultaneously)

| | Weight setting 1 | | | | | | | | | Weight setting 3 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance groups | Success ($k=1,\ldots,5$) | | | | | $\overline{Su}$ | $\overline{D}$ | Fe | Time | Success ($k=1,\ldots,5$) | | | | | $\overline{Su}$ | $\overline{D}$ | Fe | Time |
| random_30_0.5_R_$k$ | 35 | 50 | 24 | 6 | 73 | 37.6 | 1.8 | 41 | 3.3 | 48 | 46 | 22 | 10 | 67 | 38.6 | 2.6 | 69 | 2.5 |
| random_30_0.7_R_$k$ | 28 | 57 | 17 | 9 | 55 | 33.2 | 1.5 | 43 | 3.6 | 44 | 55 | 24 | 18 | 70 | 42.2 | 1.6 | 89 | 2.9 |
| random_30_0.5_C_$k$ | 47 | 97 | 92 | 98 | 96 | 86.0 | 0.2 | 92 | 3.1 | 71 | 100 | 99 | 100 | 98 | 93.6 | 0.1 | 99 | 2.6 |
| random_30_0.7_C_$k$ | 54 | 81 | 74 | 98 | 99 | 81.2 | 0.2 | 83 | 3.4 | 75 | 94 | 94 | 99 | 100 | 92.4 | 0.1 | 100 | 2.9 |
| random_50_0.5_R_$k$ | 27 | 23 | 52 | 14 | 18 | 26.8 | 3.4 | 28 | 6.3 | 40 | 20 | 67 | 15 | 16 | 31.6 | 4.8 | 50 | 5.0 |
| random_50_0.7_R_$k$ | 15 | 5 | 21 | 5 | 9 | 11.0 | 2.7 | 16 | 7.6 | 10 | 4 | 21 | 11 | 14 | 12.0 | 3.2 | 95 | 6.4 |
| random_50_0.5_C_$k$ | 99 | 69 | 95 | 91 | 70 | 84.8 | 0.3 | 88 | 5.6 | 100 | 78 | 96 | 97 | 84 | 91.0 | 0.2 | 99 | 4.7 |
| random_50_0.7_C_$k$ | 89 | 94 | 79 | 97 | 47 | 81.2 | 0.2 | 89 | 6.5 | 97 | 99 | 83 | 100 | 82 | 92.2 | 0.1 | 100 | 5.6 |
| random_100_0.5_R_$k$ | 4 | 4 | 3 | 7 | 6 | 4.8 | 7.3 | 7 | 46.6 | 3 | 5 | 13 | 8 | 14 | 8.6 | 9.2 | 36 | 38.4 |
| random_100_0.7_R_$k$ | 2 | 18 | 6 | 4 | 0 | 6.0 | 3.3 | 14 | 59.8 | 4 | 11 | 2 | 9 | 2 | 5.6 | 3.7 | 85 | 51.6 |
| random_100_0.5_C_$k$ | 53 | 55 | 89 | 75 | 75 | 69.4 | 1.2 | 72 | 34.9 | 51 | 46 | 96 | 89 | 86 | 73.6 | 1.2 | 90 | 30.2 |
| random_100_0.7_C_$k$ | 22 | 11 | 47 | 8 | 49 | 27.4 | 1.1 | 54 | 44.6 | 21 | 15 | 39 | 9 | 62 | 29.2 | 1.2 | 99 | 40.1 |

dom_*size_ratio_type_k* ($k = 1, \ldots 5$). Fig. 14 shows the shapes of random_50_0.5_$R$_1, random_50_0.7_$R$_1, random_50_0.5_$C$_1, and random_50_0.7_$C$_1, together with the perfect solution paths.

We show the results of weight setting 1 but not those of weight setting 2 because the former is simpler, and we confirmed that there was no significant difference in the results described later. In addition, we tested a different weight setting defined as $w_v = 1$ ($v \in V_b$) and $w_v = \alpha$ ($v \in V_w$) for various values of $\alpha$ ($= 2, 3, 4, 5$). We show the results obtained with $\alpha = 4$ (weight setting 3) because the most satisfactory results were obtained with this value. The purpose of this weight setting is to use the proposed GA as an approximate (heuristic) algorithm for the LPP in grid graphs, where we need to find the longest path

on the foreground subgraph because a path passing through a vertex in the white cells is infeasible. Therefore, the terms multiplied by $\alpha$ in the objective function $eval_1(y)$ (or $eval_2(y)$) play the role of a penalty function.

Table 3 shows the results for each instance group. The table lists the number of trials in which a perfect solution path was successfully found for each instance (Succeed), its average ($\overline{Su}$), the average number of cells where discrepancies occurred ($\overline{D}$), the average percentage at which the best solution path was obtained in the foreground subgraph (Fe), and the average execution time in seconds (Time).

We first focus on the results obtained with weight setting 1. As easily expected, it becomes more difficult to find a perfect solution path as the size of the input images increases. Basically, finding a perfect solution path for instances of Type $R$ tends to be more difficult than that for instances of Type $C$. The reason is probably that random images of Type $R$ contain more "holes" than those of Type $C$. $GA_{div}^{free}$ was able to find a perfect solution path at least five times for every instance of $size = 30$ and 50. Considering that the average execution time for these instances is several seconds, we would easily find an optimal solution path for most input images with sizes less than $50 \times 50$ by performing the $GA_{div}^{free}$ multiple times. For instances of $size = 100$, there were situations where $GA_{div}^{free}$ was able to reach an optimal solution path only a few times or not able to reach an optimal solution path at all, especially for instances of Type $R$. According to our observations, the population size of 100 is too small for the GA to find an optimal solution path for these instances. For example, $GA_{div}^{free}$ was able to reach an optimal solution path more efficiently by increasing the population size to 300 for these instances.

We proceed to the results obtained with weight setting 3. As expected and shown in Table 3, the use of weight setting 3 increases the average percentage at which the best solution path is obtained in the foreground subgraph. Therefore, weight setting 3 is useful when we use the proposed GA as a heuristic algorithm for the LPP in grid graphs. Fortunately, as compared with the use of weight setting 1, the use of weight setting 3 slightly improved (at least did not deteriorate) the ability to find a perfect solution path. Moreover, the average execution time of weight setting 3 was shorter than that of weight setting 1 for all instance groups. In our observation, the reason is that the search was more focused on the foreground subgraph (and optimal solution paths exist on the foreground subgraph). One may think that it is more straightforward to perform the GA on the foreground subgraph rather than on the rectangular grid graph when the GA is used as a heuristic algorithm for the LPP. According to our preliminary experiments, however, it was more effective to perform the GA on the rectangular grid graph while allowing infeasible solutions.

We have considered the case where the positions of the end points are optimized simultaneously. Finding an optimal solution under this condition is more difficult than in the case where the end points are given in advance. We briefly describe the latter case. We performed $GA_{div}^{fix}$ with weight setting 3 on the random raster images, where the end

Table 4: Results of $GA_{div}^{fix}$ on the random images (end points were given)

| | Weight setting 3 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Instance groups | Success ($k = 1, \ldots, 5$) | | | | | $\overline{Su}$ | $\overline{D}$ | Fe | Time |
| random_100_0.5_R_$k$ | 100 | 100 | 99 | 100 | 100 | 99.8 | 0.0 | 100 | 29.3 |
| random_100_0.7_R_$k$ | 100 | 89 | 100 | 86 | 100 | 95.0 | 0.1 | 100 | 38.4 |
| random_100_0.5_C_$k$ | 98 | 100 | 99 | 100 | 100 | 99.4 | 0.0 | 100 | 24.0 |
| random_100_0.7_C_$k$ | 100 | 97 | 100 | 100 | 97 | 98.8 | 0.0 | 100 | 31.3 |

points were fixed to the end points of the known perfect solution paths. The GA always reached perfect solution paths for all instances of the sizes $30 \times 30$ and $50 \times 50$. For the instances of size $100 \times 100$, Table 4 lists the result in the same form as in Table 3. The table shows that the GA reached optimal solutions with a probability of at least 86% for all instances.

The proposed GA should be compared with other algorithms for the LPP that can be applicable to general grid graphs. However, as far as we know, implementations of these algorithms were not made public. On the other hand, the LPP instances tackled by exact and heuristic algorithms [26] are available; however, these instances do not belong to the class of general grid graphs. Therefore, we did not compare our GA with other algorithms developed for the LPP. Instead, the random raster images (and the five recognizable input images) used in the experiments are provided as supplementary materials for the interested reader.

## 5 Conclusion

We proposed a GA for the PMGP. The most important contribution of this paper is the development of a crossover operator suitable for the PMGP, which is a novel adaptation of EAX for the TSP. EAX combines two solution paths with different end points (parent solutions) to generate new solution paths (offspring solutions). This allows the simultaneous optimization of the solution path and its end points. A nice property of EAX is that we can calculate the evaluation function for offspring solutions without performing the most time-consuming parts of the EAX procedure, and this allows the efficient generation of offspring solutions that improve the parent solutions. Although the framework of the GA is simple, we introduced an evaluation function for offspring solutions to maintain the population diversity in a positive manner. By combining these ideas, the proposed GA generated satisfactory solution paths for creating picture mazes in a short time (at most 17 s) for five recognizable input images with sizes up to $55 \times 105$. Moreover, we confirmed that the proposed GA can be used as a heuristic algorithm for the LPP on grid graphs, where it found optimal solutions for complicated random grid graphs of sizes up to $100 \times 100$.

## Acknowledgment

This work was supported by JSPS KAKENHI (grant number 17K00342).

## Appendix

## A Details of the random raster images

We used the SA algorithm of Ikeda and Hashimoto [6] (see Section 3.3) to generate random paths from which the random raster images used in the experiments were created. To generate random paths that satisfy the specified attribute values $size \in \{30, 50, 100\}$, $density \in \{0.5, 0.7\}$, and $type \in \{R, C\}$, we performed the SA with several evaluation functions. We were able to obtain random paths that satisfied the specified attribute values after adjusting the parameters of the SA.

When $size = n$, we searched for random paths on the $n \times n$ rectangular grid graph. For instances of Type $R$, we generated random paths by considering only the attribute value $density$; we defined the evaluation function as $|ratio - density|$, where $ratio$ is the ratio of the number of vertices passed by the current random path over the number of all vertices.

To evaluate the degree of clustering of the black cells of the random raster images, we defined a cluster level as $cl = \sum_{v \in V_b} nb(v)$, where $nb(v)$ is the number of black cells in the four neighboring cells of $v$. The average values of the cluster level for the instances of Type $R$ were 2.06 ($size = 30$, $density = 0.5$), 2.65 ($size = 30$, $density = 0.7$), 2.11 ($size = 50$, $density = 0.5$), 2.68 ($size = 50$, $density = 0.7$), 2.14 ($size = 100$, $density = 0.5$), and 2.80 ($size = 100$, $density = 0.7$) for different values of $size$ and $density$. We then determined the target values of the cluster level ($cl_{target}$) for instances of Type $C$ as $\frac{4 - cl_R}{2}$ for each configuration of $size$ and $density$, where $cl_R$ is the average of the cluster level for the five instances of Type $R$ with the same values of $size$ and $density$. We used an evaluation function defined as $|ratio - density| + \alpha\,|cl - cl_{target}|$ to generate the instances of Type $C$. The value of $\alpha$ was adjusted such that the value of this evaluation function was almost zero in the obtained local minimum solutions.

## References

[1] S. Mochizuki: Ukidashi Meiro 1, Gakken, 2006. (in Japanese)

[2] K. Yuzawa: Aiueo Meiro, Nikoli, 2003. (in Japanese)

[3] Picture This! Mazes by Conceptis Puzzles. Sterling Publishing, 2005.

[4] Y. Okamoto and R. Uehara: How to make a picturesque maze. In Proc. of the 21th Canadian Conf. on Computational Geometry, pp. 137–140, 2009.

[5] K. Hamada: A Picturesque maze generation algorithm with any given endpoints. Journal of Information Processing, 21(3):393–397, 2013.

[6] K. Ikeda and J. Hashimoto: Stochastic optimization for picturesque maze generation. Transactions of Information Processing Society of Japan, 53(6):1625–1634, 2012. (in Japanese)

[7] T. Kurokawa: Picture maze generation by successive insertion of path segment. British Journal of Mathematics & Computer Science, 4(24):3444–3463, 2014.

[8] Y. Nagata and S. Kobayashi: Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In Proc. of the 7th Int. Conf. on Genetic Algorithms, T. Bäck, (Ed.), pp. 450–457, Morgan Kaufmann, 1997.

[9] Y. Nagata and S. Kobayashi: A powerful genetic algorithm using edge assembling crossover for the traveling salesman problem. Informs Journal on Computing, 25(2):346–363, 2013.

[10] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter: Hamilton paths in grid graphs. SIAM Journal Computing, 11(4):676–686, 1982.

[11] M.R. Garey and D.S. Johnson: Computers and Intractability: A guide to the Theory of NP-Completeness. Freeman, San Francisco, 1979.

[12] G. Gutin: Finding a longest path in a complete multipartite digraph. SIAM Journal on Discrete Mathematics, 6(2):270–273, 1993.

[13] R.W. Bulterman, F.W. van der Sommen, G. Zwaan, T. Verhoeff, A.J.M. van Gasteren, and W.H.J. Feijen: On computing a longest path in a tree, Information Processing Letters, 81(2):93–96, 2002.

[14] K. Ioannidou, G.B. Mertzios, and S. Nikolopoulos: The longest path problem is polynomial on interval graphs. In Proc. of the 34th Int. Symp. on Mathematical Foundations of Computer Science, R. Královic and D. Niwinski (Eds.), pp. 403–414, Lecture Notes in Computer Science 5734, Springer-Verlag, 2009.

[15] F. Keshavarz-Kohjerdi, A. Bagheri, and A. Asgharian-Sardroud: A linear-time algorithm for the longest path problem in rectangular grid graphs. Discrete Applied Mathematics, 160:210–217, 2010.

[16] G.B. Mertzios and D.G. Corneil: A simple polynomial algorithm for the longest path problem on cocomparability graphs, SIAM Journal on Discrete Mathematics, 26(3):940–963, 2012.

[17] F. Keshavarz-Kohjerdi and A. Bagheri: An efficient parallel algorithm for the longest path problem in meshes. Journal of Supercomputing, 65(2):723–741, 2013.

[18] D. Karger, R. Motwani, and G. Ramkumar: On approximating the longest path in a graph, Algorithmica 18: 421–432, 1993.

[19] A. Björklund and T. Husfeldt, Finding a path of superlogarithmic length. SIAM Journal on Computing, 32(6):1395–1402, 2003.

[20] R. Uehara and Y. Uno: On computing longest paths in small graph classes, International Journal of Foundations of Computer Science, 18(5):911–930, 2007.

[21] H.N. Gabow: Finding paths and cycles of superpolylogarithmic length. SIAM Journal on Computing 36(6):1648-1671, 2007.

[22] Z. Zhanga and H. Li: Algorithms for long paths in graphs. Theoretical Computer Science, 377:25–34, 2007.

[23] H.N. Gabow and S. Nie: Finding long paths, cycles and circuits. In Proc. of the 19th annual International Symp. on Algorithms and Computation, S-H Hong, H. Nagamochi, and T. Fukunaga (Eds.), pp. 752–763, Lecture Notes in Computer Science 5369, Springer-Verlag, 2008.

[24] Wen-Qi Zhang and Yong-Jin Liu: Approximating the longest paths in grid graphs. Theoretical Computer Science, 412:5340–5350, 2011.

[25] M.G. Scutellà: An approximation algorithm for computing longest paths. European Journal of Operational Research, 148(3):584–590, 2003.

[26] Q. D. Pham and Y. Deville: Solving the Longest Simple Path Problem with Constraint-Based Techniques. In Proc. of Int. Conf. on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, N. Beldiceanu, N. Jussien, and É. Pinson (Eds.), pp. 292–306, Lecture Notes in Computer Science 7298, Springer-Verlag, 2012.

[27] D. Portugal, C.H. Antunes, and R. Rocha: A study of genetic algorithms for approximating the longest path in generic graphs. In Proc. of the 2010 IEEE International Conference on Systems, Man and Cybernetics, pp. 2539–2544, IEEE, Piscataway, 2010.

[28] R. Stern, S. Kiesel, R. Puzis, A. Felner, and W. Wheeler: Max is more than min: Solving maximization problems with heuristic search. In Proc. of the Seventh Annual Symposium on Combinatorial Search, S. Edelkamp and R. Bartak (Eds.), pp. 4324–4330, AAAI Press, 2014.