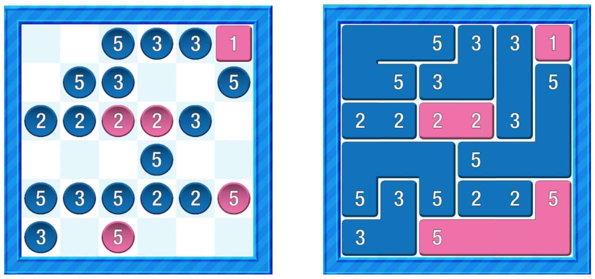
On Solving Link-a-Pix Picture Puzzles

Sanghai Guan, Jingjing Wang, *Member, IEEE*, Zhengru Fang, and Yong Ren, *Senior Member, IEEE*

***Abstract*—The Link-a-Pix puzzle, which is also known as Piczle, PathPix, Pictlink, Number Net or Paint by Pairs, is a popular picture logic puzzle game where the player paints a grid by linking the hint points with number-color labels to obtain the solution as a pixel art picture. In this letter, we propose a joint depth first searching and linear programming aided algorithm for the sake of automatically solving the Link-a-Pix puzzle. The experiments implemented on forty puzzles with various types verify the effectiveness and feasibility of our proposed solver, which is conducive to both designing and solving the Link-a-Pix puzzles and related applications.**

***Index Terms*—Link-a-Pix puzzle, Piczle, picture logic puzzle, linear programming.**



(a) The puzzle (b) The solution

Fig. 1. An example of Link-a-Pix picture puzzles.

I. Introduction

The picture logic puzzle is a kind of logic puzzles, which generates a pixel-art picture when the puzzle is solved. As a family member of picture logic puzzles, *Link-a-Pix* puzzle [1], which was originally conceived in Japan in 2002, has attracted millions of amateurs for its simple and language-independent rules, as well as artistic effects. In the past, such puzzles were termly published on puzzle magazines, while thanks to the soaring development of various touch-screen devices which provide friendly interaction and quality of experience for picture logic puzzles, game softwares designed for these intelligent platforms such as *Conceptis Link-a-Pix*, *Piczle Lines*, *Draw Puzzle* make Link-a-Pix puzzles greatly popular than ever before, and also impose a universal influence on the field of education and entertainment [2].

II. Related Works

In the literatures, the automatic solving methods for picture logic puzzles have been but not sufficiently investigated. As for the kind of linking-number puzzles, the *Numberlink* puzzle [3], which is firstly designed in 1897, is a better-known and better-studied model. Similar to Link-a-Pix puzzles, in Numberlink puzzles, the player has to link the given number pairs with simple paths to fill the grid. As a computational problem, various solvers are proposed for it and it has been proved to be an NP-complete problem [4], [5]. Comparing with Numberlink puzzles, Link-a-Pix puzzles do not assign the connecting pairs directly. Instead, it hints the connecting pairs by number-color labels, which also provides stronger constraints simultaneously. Therefore, Link-a-Pix puzzles is suitable to be formulated into a constraint optimization problem, and linear programming [6] provides a powerful tool for it. To address aforementioned issues, in this letter, we propose a linear programming based solver to form the solution by selecting the feasible links with the aid of depth first searching. This work is conducive to the solving and design of Link-a-Pix puzzles. Moreover, the study of such puzzle models can also shed light on some application problems, such as the wire routing and layout design of the very-large-scale integration [7].

Dr. Wang would like to acknowledge the financial support of the Shuimu Tsinghua Scholar Program. *(Corresponding author: Jingjing Wang.)*

The authors are with the Department of Electronic Engineering, Tsinghua University, Beijing, 100084, China (e-mail: gsh17@mails.tsinghua.edu.cn, chinaeephd@gmail.com, fangzr19@mails.tsinghua.edu.cn, reny@tsinghua.edu.cn).

III. Solver of Link-a-Pix Picture Puzzles

A. Modeling Link-a-Pix Picture Puzzles

In a common Link-a-Pix picture puzzle (as shown in Fig. 1), the player has to link the hint points with the same number-color labels in pairs on the grid, and that number equals to the number of grid points on the linking path. Moreover, every grid point must be on exactly one linking path and is dyed the color of the end points (hint points) on the path. Therefore, as the puzzle is solved, all the grid points are stained and compose a pixel art picture.

Here we assume that the puzzle grid has X rows and Y columns, respectively. The grid points with number-color labels are hint points. Each hint point can be denoted by its position $\boldsymbol{p}_i = (x_i, y_i)$. The number-color label of hint point $\boldsymbol{p}_i$ is $f(x_i, y_i) = (\mathbf{n}, \mathbf{c})$. Considering that we will not need to link the hint points with $\mathbf{n} = 1$, these grid points are removed directly from the puzzle, which compose the invalid grid point set $\mathcal{P}_0$. Furthermore, we use an $X \times Y$ matrix $\mathbf{M}_0$ to denote whether a grid point is valid or not. For example, for the puzzle in Fig. 1, we have $\mathbf{M}_0(1,6) = 0$, and the other elements in $\mathbf{M}_0$ equal to 1. Meanwhile, a vector $\boldsymbol{m}_0$ with length XY is also adopted to represent the vectorized form of $\mathbf{M}_0$. Then, the remaining P hint points that $\mathbf{n} \neq 1$ compose a set $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_P\}$, which can be divided into several subset according to their number-color labels. Here, the set of K unique number-color labels is denoted by $\mathcal{S} = \{(\mathbf{n}_1, \mathbf{c}_1), \ldots (\mathbf{n}_K, \mathbf{c}_K)\}$. In our example in Fig. 1, we have $\mathcal{S} = \{(2,$ 🟥$), (5,$ 🟥$), (2,$ 🟦$), (3,$ 🟦$), (5,$ 🟦$)\}$. Hence, we can get K subsets of hint points, i.e., $\mathcal{P}_j =$

Algorithm 1: DFS based feasible path searching

```
Function: SearchPath
Input   : current position $(x_c, y_c)$, target position
          $(x_t, y_t)$, current path $\boldsymbol{l}$, path length $d$.
1 if $d = 0$ and $(x_c, y_c) = (x_t, y_t)$ then
2     Output feasible path $\boldsymbol{l}$;
3 else if $d \neq 0$ and (1) holds then
4     for next position $(x_n, y_n) \leftarrow (x_c + 1, y_c)$,
        $(x_c - 1, y_c)$, $(x_c, y_c + 1)$, $(x_c, y_c - 1)$ do
5         if $1 \le x_n \le X$ and $1 \le y_n \le Y$ and
            $(x_n, y_n) \notin \mathcal{P}_0 \cup \mathcal{P} \cup \{(x,y) \,|\, (x,y) \in \boldsymbol{l}\} \setminus (x_t, y_t)$
            then
6             $d \leftarrow d - 1$;
7             $\boldsymbol{l} \leftarrow [\boldsymbol{l}, (x_n, y_n)]$;
8             SearchPath$\big((x_n, y_n), (x_t, y_t), \boldsymbol{l}, d\big)$;
```

$\{\boldsymbol{p}_i \mid f(x_i, y_i) = (\mathbf{n}_j, \mathbf{c}_j)\}$, where $j = 1, \ldots, K$. For the convenience of deduction, the corresponding index sets of these remaining hint points $\mathcal{I}_j^{\mathcal{P}} = \{i \mid \boldsymbol{p}_i \in \mathcal{P}_j\}$ are also constructed.

B. Searching Feasible Paths

Then, in our proposed solver, we have to find out all feasible paths between each hint point pair in each subset $\mathcal{P}_j$. Here, a feasible path is denoted by a sequence of $\mathbf{n}_j$ non-repeated grid points, i.e., $\boldsymbol{l}_i = [(x_i^1, y_i^1), \ldots, (x_i^{\mathbf{n}_j}, y_i^{\mathbf{n}_j})]$, where two end points are in the same hint point subset $\mathcal{P}_j$ with label $(\mathbf{n}_j, \mathbf{c}_j)$, and the other grid points are not labeled. The length of this path is denoted by $d = \mathbf{n}_j - 1$. Then we can find out all feasible paths between them recursively by a depth first searching (DFS) based algorithm as shown in Algorithm 1, i.e., SearchPath$\big((x_i^1, y_i^1), (x_i^{\mathbf{n}_j}, y_i^{\mathbf{n}_j}), [(x_i^1, y_i^1)], \mathbf{n}_j - 1\big)$. As searching the paths, in order to prune and improve search efficiency, a judgment of existence can be conducted according to the characteristics of grid graphs. Specifically, if there exists feasible paths with length d between (x_c, y_c) and (x_t, y_t), then there must exists:

$$\frac{d - |x_t - x_c| - |y_t - y_c|}{2} \in \mathbb{N}, \tag{1}$$

where $\mathbb{N}$ is the natural number set. It indicates that the length of any path between two grid points must equal to their Manhattan distance plus an even number $(0, 2, 4, \ldots)$.

Lastly, all the L feasible paths compose the set $\mathcal{L} = \{\boldsymbol{l}_1, \ldots, \boldsymbol{l}_L\}$, which can also be divided into subsets $\mathcal{L}_j = \{\boldsymbol{l}_i \mid f(x_i^1, y_i^1) = (\mathbf{n}_j, \mathbf{c}_j)\}$ according to the number-color labels of their end points. The corresponding index sets of $\mathcal{L}_j$ is represented by $\mathcal{I}_j^{\mathcal{L}} = \{i \mid \boldsymbol{l}_i \in \mathcal{L}_j\}$. Similarly, we also take an $X \times Y$ matrix $\mathbf{M}_i$ to represent the fill map of feasible path $\boldsymbol{l}_i$, where the grid points on $\boldsymbol{l}_i$ equal to 1, and the others equal to 0. Similarly, $\boldsymbol{m}_i$ denotes the vectorized fill map of $\mathbf{M}_i$. In practice, there may exist multiple feasible paths with the same fill map, in order to reduce the amount of optimization variables, we only keep one of them in $\mathcal{L}_j$ and delete the others.

C. Selecting Paths with Linear Programming

After obtaining all the feasible paths and corresponding fill maps, we have to select the correct paths to construct the solution. Here, the path selection problem can be formulated as an L_1 norm 0-1 optimization problem as:

$$\begin{aligned} \min \quad & \Big\| \sum_{i=1}^{L} w_i \boldsymbol{m}_i - \boldsymbol{m}_0 \Big\|_1 && \text{(2a)} \\ \text{s.t.} \quad & w_i \in \{0, 1\}, \qquad i = 1, \ldots, L && \text{(2b)} \end{aligned}$$

The optimization objective in (2a) indicates that each valid grid point must be on exactly one feasible path $\boldsymbol{l}_i$. Moreover, the optimization variable, i.e., the weight of the path $w_i \in \{0, 1\}$ in (2b) represents whether a path is selected or not. Obviously, for a solvable puzzle, the optimal value of this problem equals to 0. For the efficiency of problem solving, we relax the constrains of weight w_i and reformulate the problem into a linear programming problem, i.e.,

$$\begin{aligned} \min \quad & \mathbf{1}^{\mathrm{T}} \boldsymbol{y} && \text{(3a)} \\ \text{s.t.} \quad & -\boldsymbol{y} \preceq \boldsymbol{x} - \boldsymbol{m}_0 \preceq \boldsymbol{y} && \text{(3b)} \\ & \sum_{i=1}^{L} w_i \boldsymbol{m}_i = \boldsymbol{x} && \text{(3c)} \\ & 0 \le w_i \le 1, \qquad i = 1, \ldots, L && \text{(3d)} \end{aligned}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are auxiliary variables. For a solvable puzzle, the optimal value of above problem still equals to 0, and $w_i \in \{0, 1\}$ $(i = 1, \ldots, L)$ if the puzzle has the unique solution. However, there may exist non-integer weight w_i if the puzzle has multiple solutions. In this case, we will introduce a method in Section III-E to obtain the final solution.

D. Linear Programming for Non-Fully Filled Puzzles

In a variant of common Link-a-Pix puzzles, the grid is not designed to be fully filled. In other words, some grid points are kept blank as puzzles solved, which usually form white objects or transparent background of the solution (as shown in Fig. 3(c) and Fig. 3(d)). Apparently, such puzzles provide challenges with higher difficulty for puzzle amateurs. However, in our linear programming model, it leads to a tendency of overfilling the blank areas of the correct solution to further minimize the L_1 norm. To handle it, we add extra constraints to the original optimization problem as:

$$\begin{aligned} \min \quad & \mathbf{1}^{\mathrm{T}} \boldsymbol{y} && \text{(4a)} \\ \text{s.t.} \quad & -\boldsymbol{y} \preceq \boldsymbol{x} - \boldsymbol{m}_0 \preceq \boldsymbol{y} && \text{(4b)} \\ & \sum_{i=1}^{L} w_i \boldsymbol{m}_i = \boldsymbol{x} && \text{(4c)} \\ & 0 \le w_i \le 1, \qquad i = 1, \ldots, L && \text{(4d)} \\ & \sum_{i \in \mathcal{I}_j^{\mathcal{L}}} w_i \le \frac{1}{2} |\mathcal{I}_j^{\mathcal{P}}|, \qquad j = 1, \ldots, K && \text{(4e)} \end{aligned}$$

where $|\cdot|$ represents the number of elements of a set. The extra constraint (4e) means that for each hint point subset $\mathcal{P}_j$, the number of selected paths can not exceed half the number of hint points in $\mathcal{P}_j$. It validly restrains the tendency of

Algorithm 2: Final solution generating

```
1  Input: paths' weight $w_1, \dots w_L$, fill map $\mathbf{M}_0, \dots, \mathbf{M}_L$;
2  $w_i \leftarrow 0$ for $i$ in $\{i \mid w_i < \varepsilon\}$;
3  $w_i \leftarrow 1$ for $i$ in $\{i \mid w_i > 0.5 + \varepsilon\}$;
4  Current fill map $\mathbf{M}' \leftarrow (\mathbf{1}_{X \times Y} - \mathbf{M}_0) + \sum_{\{i|w_i=1\}} \mathbf{M}_i$;
5  Find all unfilled 4-connected regions containing
     unconnected hint points in $\mathbf{M}'$ as $\mathbf{M}'_1, \dots, \mathbf{M}'_R$;
6  for $r = 1, \dots, R$ do
7      Best filling paths' index set $\mathcal{I}_B \leftarrow \emptyset$;
8      for $t = 1, \dots, T$ do
9          Candidate paths' index set
             $\mathcal{I}_C \leftarrow \{i \mid \mathbf{M}'_r + \mathbf{M}_i \preceq \mathbf{1}_{X \times Y}, 0 < w_i < 1\}$;
10         Temporary index set $\mathcal{I}_T \leftarrow \emptyset$;
11         for $k = 1, \dots, |\mathcal{I}_C|$ do
12             Pop index $i$ randomly from $\mathcal{I}_C$;
13             if $\mathbf{M}'_r + \mathbf{M}_i + \sum_{j \in \mathcal{I}_T} \mathbf{M}_j \preceq \mathbf{1}_{X \times Y}$ then
14                 $\mathcal{I}_T \leftarrow \{\mathcal{I}_T, i\}$;
15         if $\|\sum_{i \in \mathcal{I}_T} \boldsymbol{m}_i\|_1 > \|\sum_{i \in \mathcal{I}_B} \boldsymbol{m}_i\|_1$ then
16             $\mathcal{I}_B \leftarrow \mathcal{I}_T$;
17         if all hint points in $\mathbf{M}'_r$ are connected by the
             paths in $\mathcal{I}_B$ then
18             Break;
19     $w_i \leftarrow 1$ for $i$ in $\mathcal{I}_B$;
20 Output: paths $\boldsymbol{l}_i$ with $w_i = 1$ as the final solution;
```

filling the empty grids of the solution by redundant paths. For these puzzles, the optimum value of the optimization problem equals to the number of empty grids in the solution and can be calculated in advance from the labels in the puzzle.

E. Generating the Final Solution

After solving the linear programming problem, due to the limited precision of numerical calculation, and that the puzzle may have multiple solutions, we have to handle the obtained non-integer weights and produce the final solution. Algorithm 2 describes this process. Concretely, we firstly select the paths $\boldsymbol{l}_i$ with $w_i > 0.5 + \varepsilon$ into the solution directly, and dismiss the paths $\boldsymbol{l}_i$ with $w_i < \varepsilon$, where we take a threshold $\varepsilon = 0.001$ to fix the numerical error of linear programming. Accumulating the fill maps $\mathbf{M}_i$ of these selected paths and the fill map of invalid points $(\mathbf{1}_{X \times Y} - \mathbf{M}_0)$, we can get a temporary fill map represented as $\mathbf{M}'$. The unfilled grid points in $\mathbf{M}'$ equals to 0. Then we use $\mathbf{M}'_1, \dots, \mathbf{M}'_R$ to represent the R 4-connected regions [8] in $\mathbf{M}'$ that contain unconnected hint points, where the points in given 4-connected region equal to 0 and the others equal to 1. Then, for each region, we select the candidate paths randomly and attempt to fill this region. This process is implemented repeatedly until the perfect filling is found or it reaches the maximum iteration time T. The best filling paths found in this process construct the final solution together with the previous selected paths.

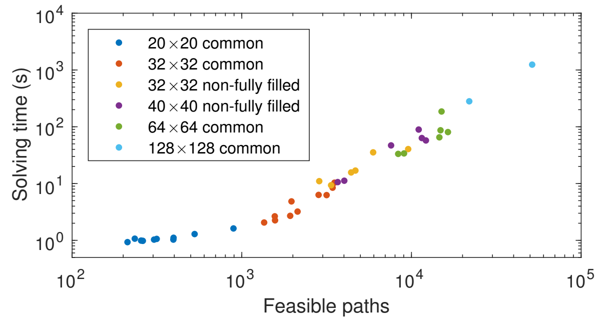


Fig. 2. The relationship between the number of feasible paths and solving time of puzzles.

IV. Experiments on Proposed Solver

In order to demonstrate the performance of proposed method, we test our solver[1] on the puzzles from a popular mobile Link-a-Pix game named *Piczle Lines DX*[2]. The solver is implemented on MATLAB and the linear programming problem is formulated and solved by CVX Toolbox [9], [10]. In the experiment, we select 40 puzzles with different themes, types and sizes. Some of the tested puzzles and obtained solution are illustrated in Fig. 3. In addition, Table I summarizes some statistics of these puzzles, including the average number of hint points and the average feasible paths found. The average solving time and accuracy are also provided. Furthermore, Fig. 2 reveals the relationship between feasible paths found in a puzzle and its solving time.

According to our test result, our solver shows robust performance and gives correct solutions for all tested puzzles. As the expansion of puzzle size, the solving time increases rapidly. The solving time and the number of feasible paths are highly correlated, and have a quasi-quadratic relationship in medium and large size puzzles. In addition, solving non-fully filled puzzles generally costs longer time than common ones. This is because these puzzles usually has higher difficulty, and the extra constraints also increase the complexity of solving the linear programming problem.

V. Conclusions

In this letter, we provide a solver for Link-a-Pix picture puzzles. Our depth first searching and linear programming based algorithm gives the precise solution with low complexity. We test our solver in the experiments based on the real puzzles of Link-a-Pix games, where we also gives further analysis on the relationship of the characteristics of the puzzle and its solving time. These results contribute to the solving and design of Link-a-Pix puzzles, as well as related application problems. In our further works, we would like to investigate the automatic design of Link-a-Pix puzzles with a given picture.

Acknowledgment

The authors are grateful to Yuncong Xie for assistance with collecting and collating the puzzle data.

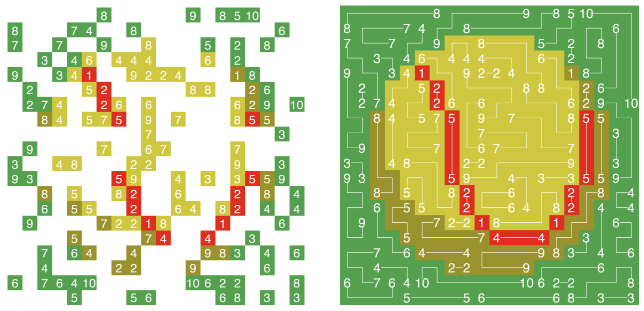
[1]The code and puzzle data are available at https://github.com/guansanghai/Link-a-Pix-Solver.

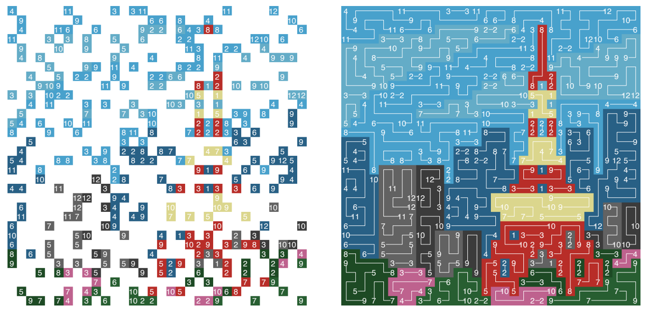
[2]http://www.piczle.club.

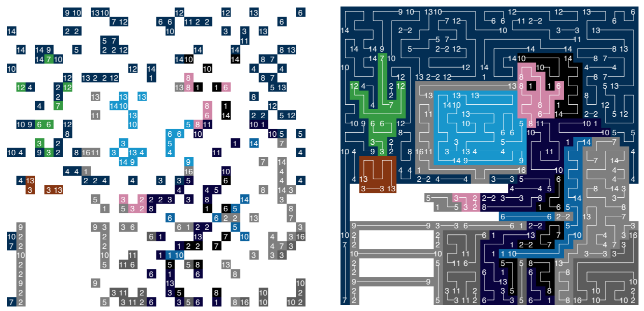
TABLE I
TEST DATA AND RESULT

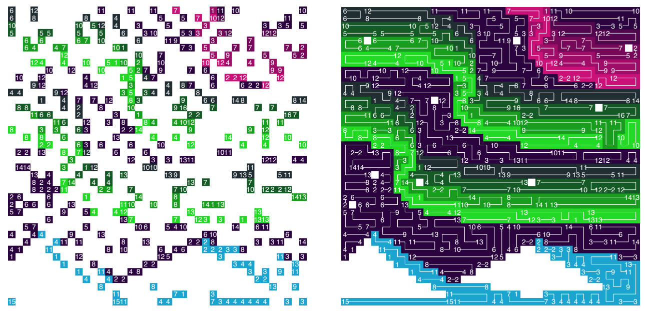
Theme	Type	# of puzzles	Size	Avg. hint points	Avg. feasible paths	Solving time (s)*	Accuracy
Sports	common	10	20×20	155.8	379.2	1.11	100%
Culture	common	10	32×32	342.9	2345.0	4.91	100%
Professions	non-fully filled	6	32×32	308.5	5134.5	21.48	100%
Nightscape	non-fully filled	6	40×40	476.3	8327.7	46.48	100%
Animals	common	6	64×64	1293.7	13043.2	80.93	100%
Dinosaurs	common	2	128×128	5333.0	36665.0	716.82	100%

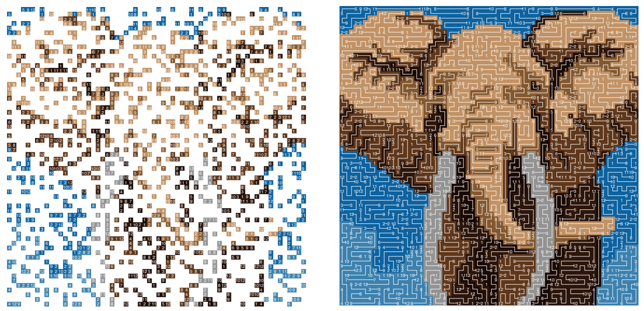
* Test environment: 2.7GHz Intel Core i5 with 8GB RAM, MATLAB R2016B, CVX v3.0 beta.


(a) 20×20 puzzle – “Tennis”


(b) 32×32 puzzle – “Tokyo Tower”


(c) 32×32 non-fully filled puzzle – “Programmer”


(d) 40×40 non-fully filled puzzle – “Aurora”


(e) 64×64 puzzle – “Elephant”

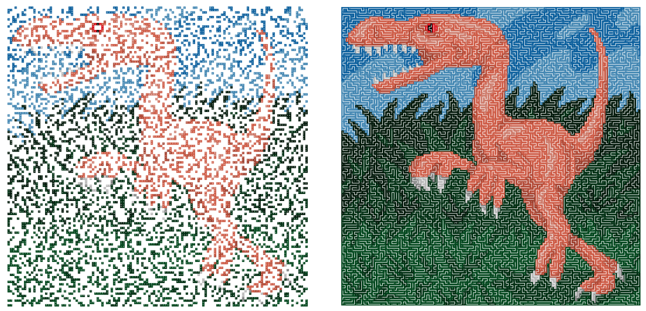

(f) 128×128 puzzle – “Velocisaurus”

Fig. 3. Some test puzzles and obtained solutions.

REFERENCES

[1] Conceptis Puzzles, “Link-a-Pix history,” 2018, [Online]. Available: http://www.conceptispuzzles.com/index.aspx?uri=puzzle/link-a-pix/history.

[2] J. Cigas and W.-J. Hsin, “Teaching proofs and algorithms in discrete mathematics with online visual logic puzzles,” *Journal on Educational Resources in Computing (JERIC)*, vol. 5, no. 2, p. 2, Jun. 2005.

[3] Wikipedia, “Numberlink,” 2018, [Online]. Available: https://en.wikipedia.org/wiki/Numberlink.

[4] R. Yoshinaka, T. Saitoh, J. Kawahara, K. Tsuruma, H. Iwashita, and S. I. Minato, “Finding all solutions and instances of numberlink and slitherlink by zdds,” *Algorithms*, vol. 5, no. 2, pp. 176–213, Jun. 2012.

[5] J. Takahashi, H. Suzuki, and T. Nishizeki, “Shortest noncrossing paths in plane graphs,” *Algorithmica*, vol. 16, no. 3, pp. 339–357, Sep. 1996.

[6] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[7] D. Richards, “Complexity of single-layer routing,” *IEEE Transactions on Computers*, vol. C-33, no. 3, pp. 286–288, Mar. 1984.

[8] D. B. West, *Introduction to Graph Theory*. Pearson, 2001, vol. 2.

[9] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming,” http://cvxr.com/cvx, Mar. 2014.

[10] ——, “Graph implementations for nonsmooth convex programs,” in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences. Springer-Verlag Limited, 2008, pp. 95–110.