A Discrete Tomography Approach to Japanese Puzzles

K.J. Batenburg [a] W.A. Kosters [b]

[a] Mathematical Institute, Universiteit Leiden, The Netherlands and CWI, Amsterdam, The Netherlands, kbatenbu@cwi.nl
[b] Leiden Institute of Advanced Computer Science, Universiteit Leiden, The Netherlands, kosters@liacs.nl

Abstract

Discrete Tomography (DT) is concerned with the reconstruction of binary images from their horizontal and vertical projections. In this paper we consider an evolutionary algorithm for computing such reconstructions. We show that the famous *Japanese puzzles* are a special case of a more general DT problem and successfully apply our algorithm to such puzzles.

1 Introduction

Discrete Tomography (DT) is concerned with the reconstruction of a discrete image from its projections. One of the key problems is the reconstruction of a binary (black-and-white) image from only two projections, horizontal and vertical (see Figure 1): for every row and column the number of black pixels is known. In 1957 Ryser [4] presented a polynomial time algorithm for finding a reconstruction — in fact, he characterized even all of them. However, the problem is usually

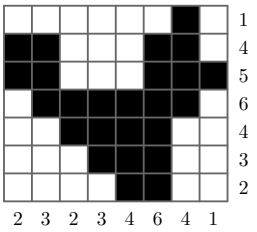

Figure 1: A binary image with its horizontal and vertical projections.

highly underdetermined and a large number of solutions may exist. In order to compute meaningful reconstructions, all available a priori information about the image (besides the projections, that can be considered as hard constraints) has to be taken into account. Suppose that we are able to define an evaluation function, which assigns a value to each of the solutions, reflecting how good a particular solution is in our context. An algorithm that maximizes the evaluation over the

set of all solutions will then yield the most desirable solution. The first author has described such an algorithm in [1] and demonstrated that it works well for several types of evaluation functions.
Japanese puzzles, also known as *nonograms*, are a form of *logic drawing*: the puzzler gradually makes a drawing on a grid, by means of logical reasoning. This task can be mimicked by using techniques from Artificial Intelligence. The solution is usually unique. Japanese puzzles are very popular in the Netherlands nowadays and are sold at every newspaper-stand.

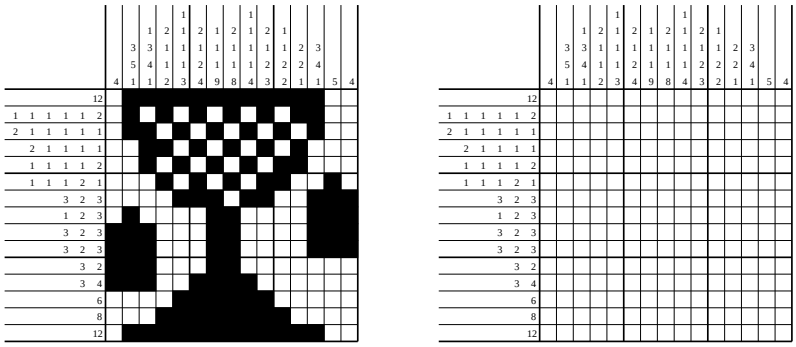

Figure 2: Left: an image and its line descriptions. Right: the corresponding Japanese puzzle.

Similar to the two-projection DT problem, the puzzler is provided with information about the horizontal and vertical arrangement of the black pixels along every line. Figure 2 shows an image and its corresponding horizontal and vertical description. For each line, the description indicates the sizes of the segments of consecutive black pixels, in the order in which they appear on the line. Summation of these sizes brings us back to the original DT problem. The problem of solving Japanese puzzles is NP-complete; several complexity results concerning Japanese puzzles, e.g., on uniqueness, were derived in [5]. Surprisingly, by choosing a suitable evaluation function, the evolutionary algorithm for discrete tomography can also be used to solve Japanese puzzles.
In the next section we will give a short description of the evolutionary algorithm. We refer to the original paper for further details. Subsequently, we show how this algorithm can be applied to Japanese puzzles.

2 Algorithm overview

We have designed a problem-specific evolutionary algorithm (cf. [3]) for solving DT problems when an evaluation function is given. The algorithm optimizes exclusively over the set of all images that satisfy the prescribed projections. At the end of each generation all candidate solutions have the prescribed horizontal and vertical projections. This requires a suitable crossover operator, that is not only capable of mixing features from both parents, but also of ensuring that the produced children

```
generate initial population P_0 of size λ, consisting of matrices in A(R,S);
perform a hillclimb operation on each image in P_0;
t := 0;
while (stop criterion is not met) do
begin
   P'_t = ∅;
   for i := 1 to μ do
   begin
      generate a child image C, by crossover or mutation;
      perform a hillclimb operation on C;
      P'_t := P'_t ∪ {C};
   end;
   select new population P_{t+1} from P_t ∪ P'_t;
   t := t + 1;
end
output the best individual found;
```

Figure 3: Outline of the evolutionary algorithm.

adhere to the prescribed projections. Similar requirements apply to the mutation operator.

Our algorithm is a *memetic algorithm* (see [2] for another view on this approach): after every crossover or mutation operation a stochastic hillclimb is performed until the solution has reached a local optimum. In this way, individuals always represent local optima in the search space.

Let m, n be the image height and width and R, S be vectors that contain the prescribed row and column projections. We denote the class of all images that have these projections by $\mathcal{A}(R, S)$. Figure 3 summarizes our algorithm. The parameters λ and μ are the population size and the number of children that are created in each generation, respectively.

An important operation often used in our algorithm is the computation of an image $X = (x_{ij}) \in \mathcal{A}(R, S)$, given R and S: the hard constraints have to be satisfied. We use a network flow approach for computing these matrices. First, we construct a directed graph N. The set V of nodes consists of a source node T_1, a sink node T_2, one layer $V_1, \ldots, V_m$ of nodes that correspond to the image rows (*row nodes*) and one layer $W_1, \ldots, W_n$ of nodes that correspond to the image columns (*column nodes*). Figure 4 shows the topology of the graph in case of a simple example. We refer to the top layer of arcs as *row arcs* and to the bottom layer as *column arcs*. Every arc in the middle layer corresponds to an entry (cell) of X, so we refer to these arcs as *cell arcs*. We assign a *capacity* to each of the arcs: every row arc is assigned the corresponding row projection, every column arc the corresponding column projection and every cell arc is assigned a capacity of 1. It is not difficult to see that a maximum integral flow from T_1 to T_2 in N corresponds directly to a solution of the tomography problem: set $x_{ij} = 1$ if arc (V_i, W_j) carries a flow of 1 and set it to 0 otherwise. Moreover, by assigning an additional set of *costs*

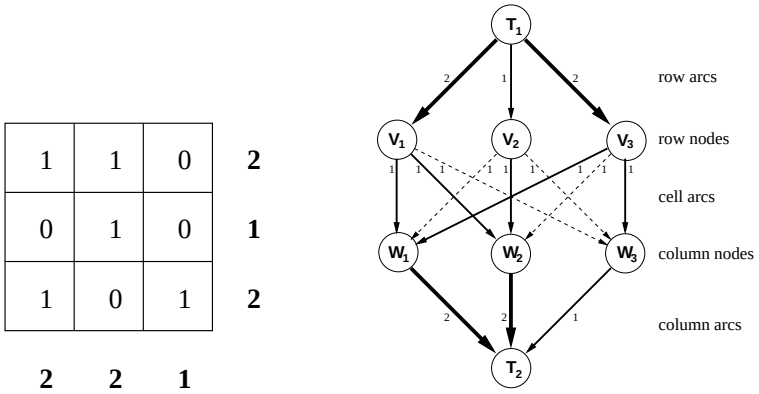

Figure 4: A 3×3 image instance and its corresponding network flow

to the cell arcs, and solving a min-cost max-flow problem in N, it is possible to express a preference among the solutions of the max-flow problem. In particular, for a given image M, we can compute in this way the matrix $X \in \mathcal{A}(R, S)$ which has the same value as M in as many cells as possible. We refer to [1] for the details of this procedure.

The crossover operator is one of the main parts of our algorithm. The input of the operator consists of two *parent images*. The output is a *child image*, which has certain features from both parents. Because all matrices in the population are members of $\mathcal{A}(R, S)$, the resulting image should have the prescribed projections.

First, a *crossover mask* $Y = (y_{ij}) \in \{0, 1\}^{m \times n}$ is computed, which determines for each pixel from which parent image it is copied. The value 0 means that the child image inherits from the first parent, 1 means that the second parent is used. The mask generation procedure is designed so that it assigns around half the pixels to each parent and so that it assigns large connected areas to each parent. In this way, local features in the parent images are often inherited as a whole by the child image.

From the crossover mask and both parent matrices $P = (p_{ij})$ and $Q = (q_{ij})$, a *model image* $M = (m_{ij})$ is computed, as follows:

$$m_{ij} = \begin{cases} p_{ij} & \text{if } x_{ij} = 0 \\ q_{ij} & \text{if } x_{ij} = 1 \end{cases}$$

Subsequently, we again use the weighted network flow model to construct a child image C, which has the same value as M in as many entries as possible. This will result in a child image that is in $\mathcal{A}(R, S)$, resembles the first parent in a certain part and resembles the other parent in the rest of the image. Figure 5 shows two parent matrices (having the same projections), a crossover mask, the corresponding model image and the resulting child image. In this example, we use the number of neighbouring pairs of black pixels as the evaluation function. Although the child image resembles both parents in their corresponding parts, it is clear that the child image is far from a local optimum with respect to the evaluation function.

To ensure that the child image has sufficient quality, we apply a local hillclimb operator after the crossover operation. Figure 6 show the basic principle of the hillclimb operator. Pairs of black and white entries are swapped as long as it is possible to improve the evaluation function in this way.
The mutation operator uses similar principles. First, a *mutation mask* is generated which determines a small part of the image that will be distorted. By using the network flow method and subsequent hillclimbing a child image is generated that adheres to the prescribed projections.

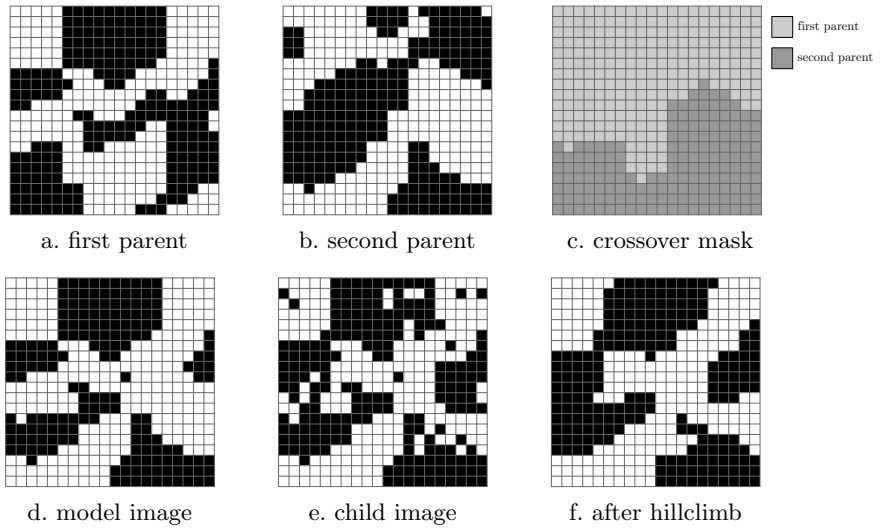

Figure 5: The crossover operator combines two parent matrices into a child image.

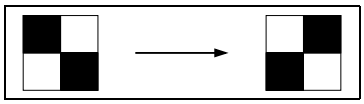

Figure 6: Swapping pairs of black and white pixels in the indicated way does not change the projections.

3 Solving Japanese puzzles

We now turn to the problem of solving Japanese puzzles. As summation of the segment sizes of a line yields the total number of black pixels on that line, Japanese puzzles can be considered to be a special form of the DT problem, in which extra a priori information is available.
We will construct an evaluation function which encapsulates all this extra information. The evaluation function should reflect the deviation of a given image from the horizontal and vertical descriptions. We consider this deviation separately for

each (horizontal or vertical) line. We can then obtain an evaluation function for the whole image by summation of the deviations of all lines. The minimal deviation possible, 0, should of course correspond to a solution of the puzzle.
We refer to the evaluation of a pixel line ℓ which has description s by $d_s(\ell)$. Ideally, the function d_s should make full use of all information available: the value of the pixels on ℓ and the prescribed description s of ℓ. Let $(\ell_1, \ell_2, \ldots, \ell_k)$ be the pixel values of ℓ (where $k = n$ if ℓ is horizontal, $k = m$ if ℓ is vertical). We call the operation of changing the value of one pixel of ℓ a *bitflip operation*. We now define $d_s(\ell)$ to be the minimal number of bitflip operations required to make ℓ conform to s.
This definition of d_s has several advantages. Firstly, it is a very intuitive way of defining the "distance" between a given line of pixels and its prescribed description and it uses all available information. If a line ℓ adheres to its description s, we have $d_s(\ell) = 0$, as desired. Secondly, d_s has the property that performing a single bitflip operation on ℓ, yielding a line ℓ', can change the value of d_s by no more than 1. In a discrete sense, d_s can be regarded as a *fluent function of* ℓ.
Surprisingly, $d_s(\ell)$ can be computed quite efficiently by means of *dynamic programming*. This is of course necessary since the algorithm requires many of these computations. Suppose that the prescribed description s of ℓ consists of h segments of black pixels, $(s_1, \ldots, s_h)$. Without loss of generality, we may assume that ℓ starts and ends with a white pixel: adding white pixels at the beginning or end of a line does not change its description.
We define a *line segmentation* $\hat{\ell} = (\hat{\ell}_1, \ldots, \hat{\ell}_{2h+1})$, corresponding to the number h of black segments in s:

$$\hat{\ell}_i = \begin{cases} 0 & \text{(white)} \quad \text{if } i \text{ is odd} \\ 1 & \text{(black)} \quad \text{if } i \text{ is even} \end{cases}$$

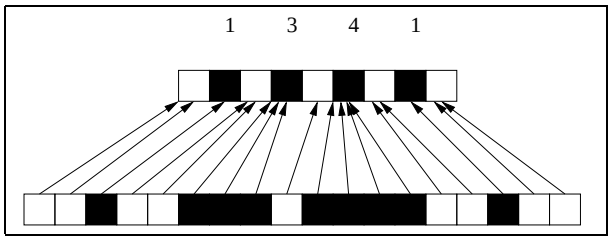

Figure 7: A line description (top), the corresponding line segmentation (middle), and an actual line that adheres to the description (bottom). The arrows indicate the links between the line and the segmentation.

The entries of the line segmentation correspond to the alternating black and white segments in the description (where we consider the white segments to be implicitly present). Figure 7 shows a line description s, its corresponding line segmentation $\hat{\ell}$ and a *realisation of* s, which is a line ℓ that adheres to s. The arrows indicate for each pixel of ℓ to which entry of $\hat{\ell}$ it corresponds. If pixel i corresponds to entry j

of $\hat{\ell}$, we say that pixel i is *linked* to entry j of the line segmentation. We call the corresponding mapping a *link mapping*.
If ℓ does not correspond to s, we can also link every pixel i consecutively to an entry j of $\hat{\ell}$. In that case, however, there are pixels in ℓ that do not match the colour of the entries of $\hat{\ell}$ to which they are linked. Some bitflip operation will have to be applied to these pixels in order to make ℓ conform to $\hat{\ell}$, with the given links.
Valid link mappings must satisfy several requirements: consecutive pixels should be linked either to the same entry of $\hat{\ell}$ or to consecutive entries of $\hat{\ell}$. The number of pixels linked to each black entry $\hat{\ell}_j$ must be $s_{j/2}$. There must be at least one pixel linked to each white entry of $\hat{\ell}$.
A *valid partial link mapping from* $(\ell_1, \ldots, \ell_i)$ *to* $(\hat{\ell}_1, \ldots, \hat{\ell}_j)$ satisfies the same requirements as a total valid link mapping. In addition, if pixel i is linked to a black entry $\hat{\ell}_j$, the pixels $(\ell_{i-s_{j/2}+1}, \ldots, \ell_{i-1})$ should also be linked to entry j.
We now introduce $\delta(i, j)$ (for $1 \leq i \leq k, 1 \leq j \leq 2h+1$), the minimal number of bitflip operations, over all valid partial link mappings from $(\ell_1, \ldots, \ell_i)$ to $(\hat{\ell}_1, \ldots, \hat{\ell}_j)$, that has to be applied to ℓ in order to make it conform to the partial link mapping (so that all pixels have the same colours as the entries to which they are linked). We remark that $\delta(k, 2h+1)$ is the minimal number of bitflip operations that is required to transform ℓ into a line that adheres to s. We can directly use it as the evaluation function for our algorithm. Fortunately, $\delta(i, j)$ can be computed efficiently by means of dynamic programming. For both the case that j is odd and the case that j is even, it is possible to construct a recurrence relation which computes $\delta(i, j)$ from other values $\delta(i', j')$ where always $i' < i$. By using a nested loop, which iterates over all pairs of (i, j) in the right order, each value $\delta(i, j)$ can be computed from table-values that are already known.

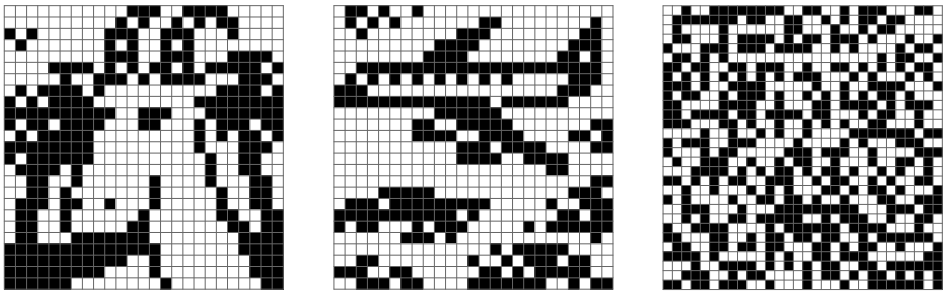

Figure 8: Test images for the Japanese puzzle variant of the algorithm.

4 Results and Conclusions

We implemented the evaluation function and performed several test runs. We used $\lambda = 1000$, $\mu = 500$ for the experiments. Figure 8 shows the test images that we used. The first two examples, of size 25×25, are from the Dutch "Puzzelsport" series and are known to have a unique solution. We performed one test run for each

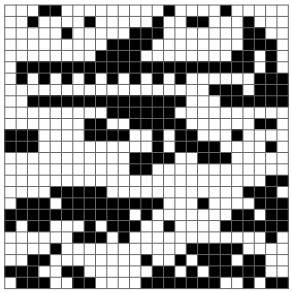

Figure 9: Reconstruction result for the second test image.

image. The first image was reconstructed perfectly on a Pentium IV at 2.4 GHz in 80 minutes after 15 generations of the algorithm. The second image proved to be much harder: the algorithm converged to a local optimum. The resulting (incorrect) reconstruction is shown in Figure 9. Note that the reconstruction is quite different from the original image, yet it adheres to nearly all line descriptions.
Surprisingly, the third test image, a random 30×30 image (50% black), was reconstructed perfectly in about 12 hours. Although the reconstruction process took a long time to complete, this is still a very positive result, since random images are very hard to reconstruct using only logic reasoning. Branching seems inevitable for that type of image.
Future research includes a detailed analysis of the performance on puzzles of different difficulty levels and different types (e.g., more than two colours).
The fact that the evolutionary algorithm from [1] can also be used to solve Japanese puzzles, which it was not specifically designed for, clearly demonstrates its versatility.

References

[1] K.J. Batenburg. An evolutionary algorithm for discrete tomography. *Discrete Applied Mathematics* (to appear), 2004.

[2] K.J. Batenburg and W.J. Palenstijn. A new exam timetabling algorithm. In T. Heskes, P. Lucas, L. Vuurpijl, and W. Wiegerinck, editors, *Proceedings of BNAIC 2003, the Fifteenth Belgium-Netherlands Artificial Intelligence Conference, Nijmegen, The Netherlands*, pages 19–26, 2003.

[3] Z. Michalewicz. *Genetic Programs + Data Structures = Evolution Programs.* Springer-Verlag, third edition, 1996.

[4] H.J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957.

[5] N. Ueda and T. Nagao. NP-completeness results for nonogram via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996.