A Peg Solitaire Font

A PEG SOLITAIRE FONT

Taishi Oikawa
National Institute of Technology, Ichonoseki College
Takanashi, Hagisho, Ichinoseki-shi 021-8511, Japan.
a16606@g.ichinoseki.ac.jp

Kazuaki Yamazaki
Japan Advanced Institute of Science and Technology
Asahidai, Nomi, Ishikawa 923-1292, Japan.
torus711@jaist.ac.jp

Tomoko Taniguchi
Japan Advanced Institute of Science and Technology
Asahidai, Nomi, Ishikawa 923-1292, Japan.
tomoko-t@jaist.ac.jp

Ryuhei Uehara
Japan Advanced Institute of Science and Technology
Asahidai, Nomi, Ishikawa 923-1292, Japan.
uehara@jaist.ac.jp

Abstract

Peg solitaire is one of the most popular classic puzzles around the world. It was proved that this puzzle was computationally intractable in general in 1990. The most common form of the puzzle consists of board with 33 holes and 32 pegs. A lot of solutions have been found by puzzle players by hand, and heuristic algorithms were developed in the 1990s. However, (super)computers running sophisticated algorithms can now enumerate all the solutions for this puzzle in a few minutes. That is, we can now completely solve certain peg solitaire puzzles of reasonable size. Using this technique, we design a "*peg solitaire font*" in the following way. We start with a peg solitaire puzzle on a board of size 5×7, which consists of 35 holes filled using 34 pegs placed in all holes except the central hole. Our algorithm running on a (super)computer generates all possible patterns reachable from the initial state. We find that there are 1,045,173,439 reachable patterns from the initial state. From these reachable patterns, we extract or "design" our font so that each of the characters in our font can be reached from the initial state. Readers are invited to solve the associated peg solitaire puzzle for each character.

Introduction

In this paper, we investigate a well-known board game for one player named "peg solitaire." According to Martin Gardner, "Worthwhile or not, no other puzzle game played on a board with counters has enjoyed such a long, uninterrupted run of popularity as solitaire" [5, Chapter 11]. Although there are some variants of the board, the most popular board consists of 33 holes as in Figure 1. The game starts with an initial state; we begin with 32 pegs on the all 33 holes except one (center) hole. The objective is to make a series of jumps that will remove all pegs but one. The goal state is to leave one peg on the hole, which was empty in the initial state. A jump consists of moving a peg over any adjacent peg to land on the next vacant hole. The jumped peg is taken off the board. No diagonal jumps are allowed. (See Figure 2.)

For this puzzle, there are tons of research from the viewpoint of recreational mathematics (see [5, 1] for further details).

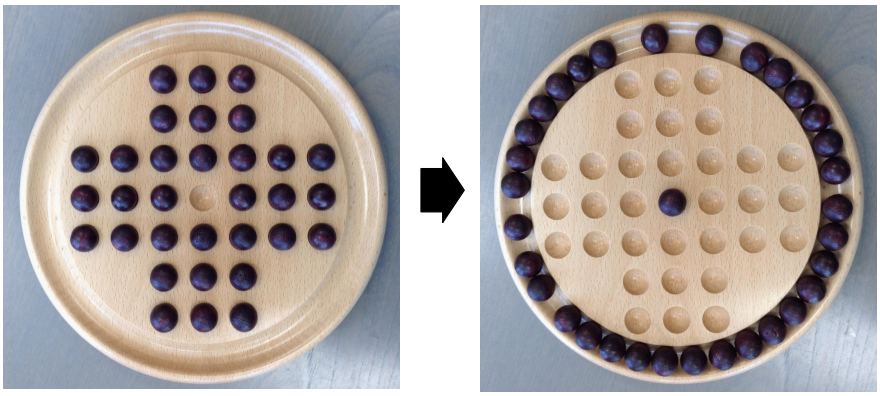


Figure 1 : *The initial and goal states of the most popular peg solitaire.*

Figure 2 : *Jump operation*

From the computational point of view, Uehara and Iwata proved that this puzzle was NP-complete in general in 1990 [9], which means that this puzzle is intractable even for a supercomputer in general form. Beside that, Uehara succeeded in finding some solutions to the puzzle by using a computer in his master thesis in 1991 [8]. In his thesis, he developed a heuristic search in the search tree of the puzzle. His program ran more than a week, and struggled to find some solutions. In 1991, he said that "it is impossible to find all solutions in this search tree." Martin Gardner also said that "No one knows how many different ways there are to solve the puzzle leaving the last counter in the center" in [5].

Since then, after a quarter of a century, peg solitaire of this size became tractable. In 2009, George Bell computed the number of solutions of the puzzle in Figure 1 in 7 minutes [2], and recently, Kanemoto, Saitoh, Kiyomi, and Uehara confirmed it in 100 seconds [6]; in total, there are 40,861,647,040,079,968$\sim 4.1 \times 10^{16}$ solutions.

Based on the algorithm, we now turn to an artistic application of peg solitaire. We design a font on the framework of peg solitaire. On his web page [3], Demaine maintains mathematical and puzzle fonts/typefaces. Our work is inspired by his invited talk presented at JCDCGGG 2016 in Tokyo [4]. In order to design a nice font, first, we change the board to a rectangle of size 5×7 as shown in Figure 3.

On the initial state, we put 34 pegs except the central hole. From this initial state, we used our algorithm on a supercomputer to compute all reachable patterns. We obtained $1,045,173,439 \sim 1.0 \times 10^9$ different patterns in 28 minutes. Next, we selected "nice" characters from these patterns. In Figure 3, we show two of them; they are "A" and "Z" reachable from the initial state. We also provide animations in GIF format for each character. On the website [7], the solutions to user-entered words are shown.

In this paper, we show the details of efficient algorithms and the other characters generated by the algorithms.

Algorithm

When we design a font, we may use trial and error to fix the shape of the characters. We have to restrict ourselves to selecting characters that represent reachable patterns from the initial state. Moreover, after obtaining a font, we like to make animated GIFs that display the sequence of jump operations to show that each character is really reachable from the initial state. In order to do that, we develop three programs.

1. Puzzle Solver: First, we develop a puzzle solver. Since the basic idea is similar to one can be found in [6], we give it briefly. Each arrangement of pegs can be represented by a binary number of 35 digits. For example, the initial state can be represented by "11111 11111 11111 11011 11111 11111 11111." We use

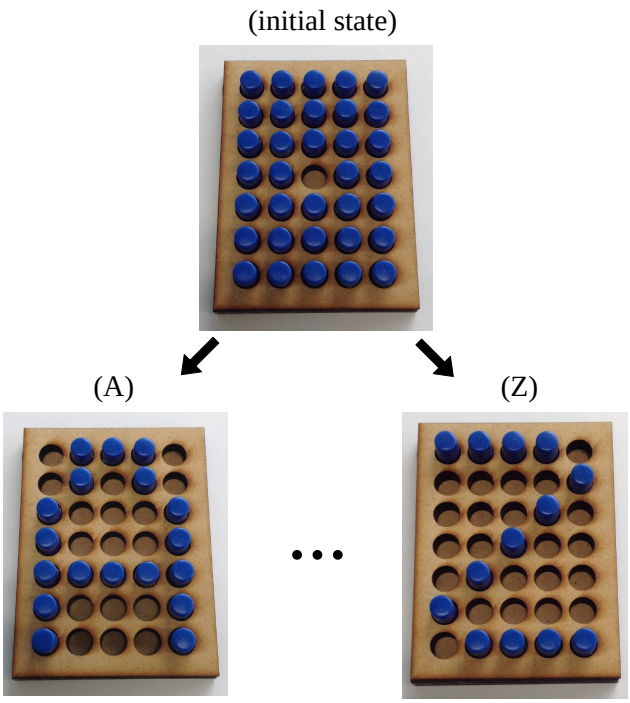


Figure 3 : *Our initial state and final states of "A" and "Z."*

this "binary number" as an index of a huge array, say, $b[]$. In [6], $b[i] = 1$ means this state i (in binary number of 35 digits) is reachable from the initial state, and $b[i] = 0$ means this state i is not reachable from the initial state (or not yet checked). In our case, we keep more information to make animated GIFs. Here, we mention that there are 21 possible horizontal jumps and 25 vertical jumps. For each of these 46 jumps, we assign its unique identifier from 2 to 59 (the detail of this rule is omitted). Then our array $b[]$ keeps the information as follows; (0) "$b[i] = 0$" means "not yet checked", (1) "$b[i] = 1$" means "this is the initial state", and (2) "$b[i] = k$" means "the last jump to this state i has identifier k". That is, each $b[i]$ takes integer from 0 to 59 (which requires 6 bits).

We note that after t jumps, the number of pegs is $34 - t$. Moreover, we confirmed that this initial state cannot reach to the state with one peg. Therefore, we need at most 32 jumps to find any arrangement from the initial state.

In order to simplify the program, we prepare two arrays of odd and $even$ to store the information. Then the algorithm can be described as follows;

0. initialize $even[i] = 0$ for all i but $i' = 11111111111111110111111111111111111$, and $even[i'] = 1$;

1. repeat the following steps 2-3 16 times;

  2. for each i with $even[i] > 0$, do the following steps;

    2.1. output i with $even[i]$;

    2.2. for each possible jump k and resulting next arrangement j, set $odd[j] = k$;

    2.3. set $even[i] = 0$ (to discard redundant output);

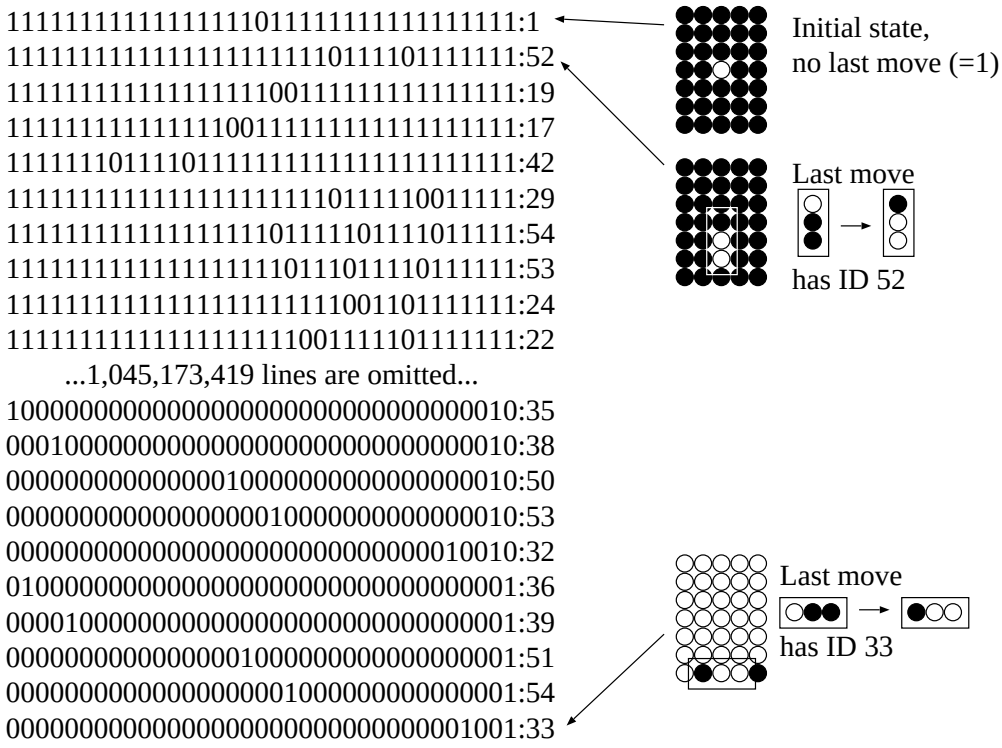
  3. for each i with $odd[i] > 0$, do the following steps;

    3.1. output i with $odd[i]$;

    3.2. for each possible jump k and resulting next arrangement j, set $even[j] = k$;

    3.3. set $odd[i] = 0$ (to discard redundant output);

In order to simplify the program, we use 8 bits (= 1 byte) to store each element of $odd[]$ and $even[]$. Therefore, in total, the arrays require 2×2^{35} bytes, which equal to 64 gigabytes. In order to deal with huge data in memory, we use a supercomputer SGI UV3000 in JAIST. It takes 28 minutes to output all reachable patterns from the initial state. As a result, we obtain 1,045,173,439 lines of reachable patterns, which are as follows:



Note: From the viewpoint of theoretical computer science, the above data can be compressed using trie, or prefix tree; then the following process can be performed much faster with much smaller memory.

2. Character Finder: We developed a simple program to do this step. For a given font character pattern D and distance d, the program checks if the reachable patterns contain ones of Hamming distance at most d from D, where *Hamming distance* is defined to be the number of bits where two patterns differ. For each D and d, this is easy to check it in a few minutes. Our program can check two or more choices for a given character's design in parallel. We repeated the searching process for each character until we achieved a nice font.

3. GIF Animator: For any given font character pattern D in the reachable patterns, traversing in the output of Puzzle Solver program backward, we can reconstruct a solution to the pattern. More precisely, this traverse can be achieved as follows:

First, find D in the output. That is, if D is a reachable pattern, the output contains the form $\hat{D} : k$, where $\hat{D}$ is D in binary representation, and k is the identifier of the jumping operation. Then we rewind the jumping operation with identifier k, and obtain a previous state D'. Next, repeat this process until the jumping operation is $k = 1$, which means that the current state is the initial state. Finally, we come to a solution in reverse ordering.

We note that for a given pattern D, we have only one solution. In general, there are many ways to reach D from the initial state, but our algorithm does not pay attention to it.

Designed Font

Among 1,045,173,439 reachable patterns from the initial state, we designed a set of peg solitaire font in Table 1.

Acknowledgements

The authors thank Hiro Ito for his great suggestion to design a font in our framework when [6] was presented.

References

[1] John D. Beasley. *The Ins and Outs of Peg Solitaire*. Oxford University Press, 1992.

[2] George I. Bell. Notes on solving and playing peg solitaire on a computer. arXiv:0903.3696, Mar. 2009.

[3] Erik D. Demaine. Mathematical and Puzzle Fonts/Typefaces. `http://erikdemaine.org/fonts` (available on April, 2017.)

[4] Erik D. Demaine. Fun with Fonts: Algorithmic Typography. See arXiv:1404.1775, September 2016.

[5] Martin Gardner. *Knots and Borromean Rings, Rep-Tiles, and Eight Queens: Martin Gardner's Unexpected Hanging*. Cambridge University Press, 2014.

[6] Itsuki Kanemoto, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara. Counting the Number of Solutions for Peg Solitaire. COMP2016-14 Vol. 116, No. 211, pp. 1-5, IEICE, 2016.

[7] Ryuhei Uehara. Peg Solitaire Font 5×7. `http://www.jaist.ac.jp/~uehara/fonts/peg-solitaire/` (available on April, 2017.)

[8] Ryuhei Uehara. *Research on analysis of a one-player game*. Master thesis, University of Electro-Communications, 1991.

[9] Ryuhei Uehara and Shigeki Iwata. Generalized Hi-Q is NP-Complete. *The Transactions of the IEICE*, E73(2):270–273, 1990.

Table 1 : *Our peg solitaire font.*