Use Of Jigsaw Puzzle Solving Algorithms In The Real World

Luca Sartore

May 2023

Contents

1 Abstract

The jigsaw puzzle problem has been in the eye of computer scientists for a while, and some clever solutions have already been found. These algorithms are made to work with a "digital" jigsaw puzzle F1, but there aren't papers (at least not popular enough to be searchable) that try to apply the solution to a "real world" jigsaw puzzle F2.

The problem has been tackled by some small projects. But, as said earlier, the process and eventual challenges has never been documented by a full paper, this wants to be the first.

As a bonus the paper will also cover the creation of a user friendly app that will be open source and free to use.

2 Introduction

2.1 Classification

This paper will focus on type 2 puzzles. A type 2 puzzle is a puzzle where the position, and the orientation of each piece is unknown.

2.2 Digital vs Real-World Jigsaw Puzzles

There is another important distinction between different types of puzzles. They can be divided into "digital" and "real world" jigsaw puzzles.

Figure F1: An example of a "digital" jigsaw puzzle
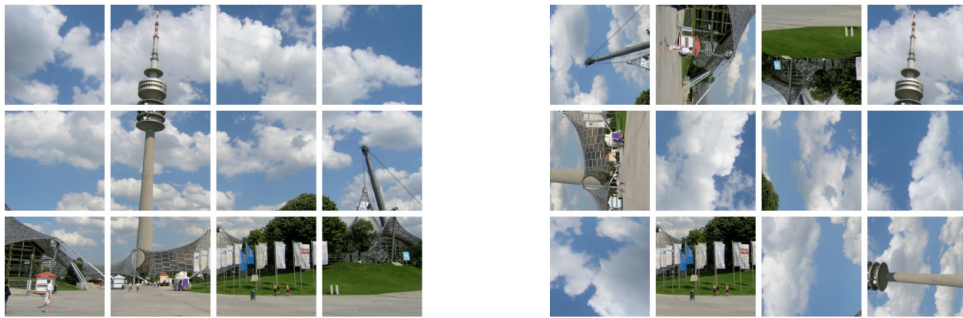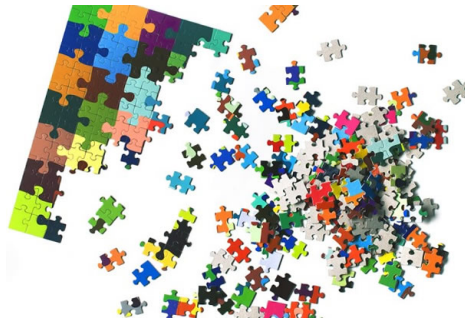

Figure F2: An example of a "real world" jigsaw puzzle

The reason this distinction is important is because, despite the generic concept of the puzzle not changing, obtaining accurate matches of a piece's characteristics is far easier with a digital puzzle, since there are far less things that can go wrong.

Figure F3: An example of what can go wrong when dealing with the real world

3 Previous Literature

This section will analyze 3 different algorithms that have been proposed as a solution of type 2 puzzles. The objective is to understand the strengths and the weaknesses of each one, to build up some knowledge that will be useful for the next section.

3.1 General Structure Of The Algorithms

All the algorithms that will be analyzed are composed of 3 sub algorithms:

3.1.1 The Splitter:

This component takes as input one or more images containing all the pieces. It then split all the pieces from each other, 9and split each piece into his four sides.

3.1.2 The Comparator:

This component compares each side with all the others, in order to understand whether they match or not.

There are two distinct kinds of "Comparator" algorithms: The "Binary Comparison" and the "Non Binary Comparison". As the name suggests when comparing two sides with a "Binary Comparison" the result can either be 0 (they do not match) or 1 (they match). In contrast a "Non Binary Comparison" can give any value between 0 and 1. This allows states of uncertainty to be represented.

3.1.3 The Solver:

This component uses the information provided by the Comparator 3.1.2, and tries to find a solution (i.e. a position and an orientation for each piece) that is the most likely to be correct.

3.2 Solving Jigsaw Puzzles By The Graph Connection Laplacian [1]

This algorithm falls in the "binary comparison" category 3.1.2. And can be used to understand the strength and weaknesses of this approach.

The main benefit of this approach is speed, having a binary comparison allows some specific optimization, in particular the use of some graph theory techniques.

The algorithm has a time complexity of ruffly $O(N^2)$. Which is in line with the theoretical minimum for jigsaw puzzle solving, according to the study: "No easy puzzles: Hardness results for jigsaw puzzles [3]".

The negative aspect might be the accuracy, since by using only two states (match or not match) some informations are lost, compared to a non binary comparison. To quantify accuracy the paper used the "neighbors comparison" metric, which "calculates the percentage of pairs of image patches that are matched correctly".

3.3 A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles [2]

This paper applies a genetic algorithm to the jigsaw puzzle problem, with some promising results.
The algorithm seems to have a $O(N^2)$ time complexity, with time slightly lower than the previous example (But it is impossible to know for sure which one is faster, given that they didn't specify the hardware used). The paper has used the same method to evaluate accuracy that the previous one used (neighbors comparison) so it is easy to compare them.
The algorithm falls in the "Non binary comparison" 3.1.2; this should give it an advantage, since it has more data to work with.
Unfortunately this advantage does not compensate for the worst precision of the algorithm itself, and the accuracy results are equivalent, if not slightly worse than the previous algorithm.
It is important to keep in mind that for the very nature of genetic algorithms, it might be possible that the accuracy would have been better if they had allowed it to run for more generations.

3.4 Computer Vision Powers Automatic Jigsaw Puzzle Solver [4]

The solutions 3.2 and 3.3 can be considered "state of the art" for digital jigsaw puzzles. But, as already introduced in section 2.2 and better explained in section TODO, appalling the solution to the real world can be challenging.
This solution is possibly a "state of the art" for real world puzzles.
In reality calling this a "solution" is incorrect, since all it does is find the best 8 matches and asks the user which one to place. Making the algorithm not autonomous.
Even ignoring this major detail, making an analysis of this solution has proven to be challenging, since the article is more of a publicity stunt than an academic paper. In fact, the code is not open source and the information about the time needed to solve the puzzle is missing.
The article only shows a single puzzle solved, with a 9×6 dimension.
Even with all of these problems this article still is one of the best example of a solution for real world puzzle.

4 The Objective Of This Paper

After reading some of the previous literature for this problem, some conclusions could be drawn: The problem of "digital jigsaw puzzle" has been solved many times, and is already well optimized.
On the other hand, the problem of "real world jigsaw puzzle" has mostly been ignored by the academic community.

The objective of this paper is to fill this void, by answering the following questions:

- Can the current state of the art solutions for digital puzzles be used for real world puzzles, or the unpredictable nature of real world data makes it impossible?

- If the algorithms cannot be applied to the real world, what is a possible solution that is more redundant to imprecise data?

5 The Setup

To answer these questions, some data to start with, and an algorithm to analyze the data; in particular the necessary components are:

5.1 The Puzzle To Solve

The first step is finding a real world puzzle to solve; It has been decided to use a classic 500 piece puzzle, and use a printer's scanner to digitize the pieces.
The puzzle has been scanned in various sub-sizes to allow the testing of different configurations. A resolution

of 1200 ppi has been used to ensure a sharp and precise image of the piece, as seen in image F3. The puzzle has been scanned backward, since the comparator algorithm that has been used REF does not use colors, and having an uniform color would make splitting the pieces easier.

5.2 Splitting The Pieces

As introduced in section 3.1.1, the first step to solve a puzzle is to split an image composed of many pieces into individual ones. This is relatively straightforward, and can be done with the following steps:

- Apply a threshold and transform an RGB image into a binary mask.

- Find all individual Blobs (a Blob is just set of connected pixels with the same color) and for each one:
    - Calculate the area (the number of pixels the Blob is made of)
    - Add it to the set of pieces only if his area in within a certain range (to avoid small pieces of dust ending up counted as pieces)

5.3 Splitting The Sides

The next step following the general template 3.1.1 is to split a pieces into his 4 sides, this could be done with the following steps:

- **Remove the holes from the piece:**
  To do so the program computes the convex hull of the piece, then calculates the difference between the convex hull and the original image, and obtains an image with the "filler areas". These filler areas have very different shapes, depending on what has generated them (a "hole" or a "knob" in the piece). So they can be classified, and added to the original piece only if they are generated by a hole, obtaining an image without holes.

- **Remove the knobs from he piece**
  To do so the image without holes gets eroded and then dilated. This generates an image that resembles the original one, but has no knobs. Then the program finds the pixels that are white in the image without holes, but not in the image that has been eroded and dilated. This generates a set of Blobs, but again, the shape of the blob is very different depending on what has generated it (an "angle" or a "knob"), so they can be classified. Then the program removes from the image with no holes all the pixels that are close to a Blob generated by a knob, obtaining an image without knobs and holes.

- **Find the coordinates of the four corners**
  To find the coordinates of the corners, the program uses a tree steps approximation. It initially finds the minimum enclosing rectangle, and uses the coordinates of the corners of the rectangle as first approximation.
  As a second approximation it takes the white pixel that is closest to the first approximation. Then it considers only the pixels that are within a certain distance of the second approximation, and finds the minimum enclosing triangle of sed pixels.
  The corner of the triangle that is closest to the second approximation is the third and final approximation of the corner of the piece.

- **Split each side using the previously found coordinates.**
  Once the program has found the four coordinates of the corners, to obtain each side,it just needs to rotate and crop the piece, and so obtain a single side.
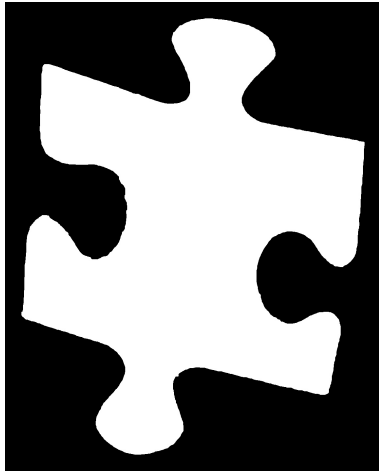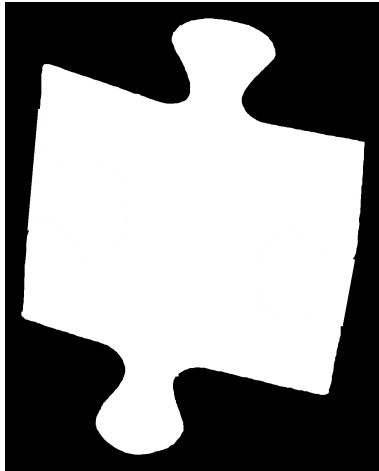
(a) Original binary mask of a piece.
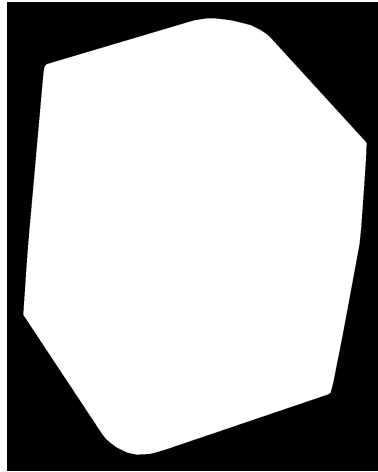
(b) Convex Hull of a piece.
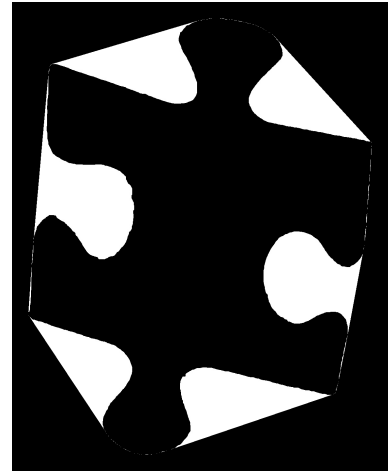
(c) Difference between (b) and (a).

(d) Piece (a) with holes removed.

(e) Mask (d) eroded.

(f) Mask (e) dilated.

Figure F4: Steps needed to split an image into his four sides.

References

[1] Vahan Huroyan, Gilad Lerman and Hau-Tieng Wu, Solving Jigsaw Puzzles By The Graph Connection Laplacian, 2020. https://arxiv.org/pdf/1811.03188.pdf.

[2] Dror Sholomon, Omid David and Nathan S. Netanyahu, A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles, 2013. https://openaccess.thecvf.com/content_cvpr_2013/papers/Sholomon_A_Genetic_Algorithm-Based_2013_CVPR_paper.pdf.

[3] Michael Brand, No easy puzzles: Hardness results for jigsaw puzzles, 2015. https://www.sciencedirect.com/science/article/pii/S0304397515001607.

[4] AbtoSoftware, Computer Vision Powers Automatic Jigsaw Puzzle Solver, 2019. https://www.abtosoftware.com/blog/computer-vision-powers-automatic-jigsaw-puzzle-solver.