CONSTRUCTING THE TOPOLOGICAL SOLUTION OF JIGSAW PUZZLES

J. De Bock, P. De Smet, W. Philips and J. D'Haeyer

Dep. TELIN/TW07, Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
E-mail: jdebock@telin.UGent.be

ABSTRACT

In this paper we present a novel approach to the jigsaw puzzle solving problem. The main components are a shape based local contour matching followed by a global solving procedure that constructs the topological solution of the jigsaw puzzle. The shape based local contour matching will be discussed briefly, but the main focus of the paper is the construction of the topological solution. The solving procedure starts with the classification of the puzzle pieces. Next, the edge topology is constructed and finally the internal topology is constructed. We tested the developed algorithms on five different jigsaw puzzles. Ultimately, we were able to solve a jigsaw puzzle consisting of 300 puzzle pieces, the largest one solved automatically to date.

1. INTRODUCTION

Automatic jigsaw puzzle solving has always been a popular testbed for the evaluation of computer vision techniques. One has to find a solution for three very different subproblems: local matching (geometric), finding a global solution (combinatorial) and explicitly fitting the pieces together (geometric). You can regard it as a stripped down version of the very complex generic digital reconstruction problem. The clearly defined and easy understandable goal appealed to the imagination of several authors in the past [1, 2, 3, 4] and also very recently [5, 6].

The first step in automatically solving a jigsaw puzzle by computer, is obtaining a digital image of the individual puzzle pieces. This is done by scanning the pieces with a flatbed image scanner. Using real jigsaw puzzles is to be preferred above making artificial jigsaw puzzles [1]. When you use real jigsaw puzzles, you can test the developed algorithms for robustness against scanner noise, dust particles and bad jigsaw cuttings. The pieces are also scanned with random rotations. Limiting the rotation to predefined angles is too restrictive and unrealistic, because generally you do not know which rotation the piece will have in the final solution.

Starting from the digital images of the individual pieces, we apply different algorithms to obtain a good description of the local shape along the contour of the pieces. Those algorithm are already published in [7, 8]. We briefly enumerate the different steps: contour extraction with region growing, polygonal approximation of the contour and a shape description based on differential angles. The final result is, for each piece, a shape vector. The typical contour outlines of the pieces are shown in figure 1.

We then apply a contour matching algorithm on the shape vectors to find the cost and positions of the best match between a pair of pieces. In [7, 8], the same algorithms were used to search for matching fragments of ripped-up documents with success. This proves that the matching algorithm we use is not specific for puzzle pieces, unlike other papers [4, 5, 6]. The contour matching costs will be the only input for the algorithms that construct the topological solution. We do not mix the construction of the topological solution with the last step, i.e. explicitly fitting the pieces together, as in [5, 6]. We thus first construct the topological solution and then we can make a graphical solution with the algorithms published in [9]. The main advantage of this strict distinction is that the time consuming explicit fitting is not multiplied with the many pairs of probably matching pieces that must be tried during the solving step. This approach is also taken in [3]. In the following sections the construction of the topological solution will be explained in detail.

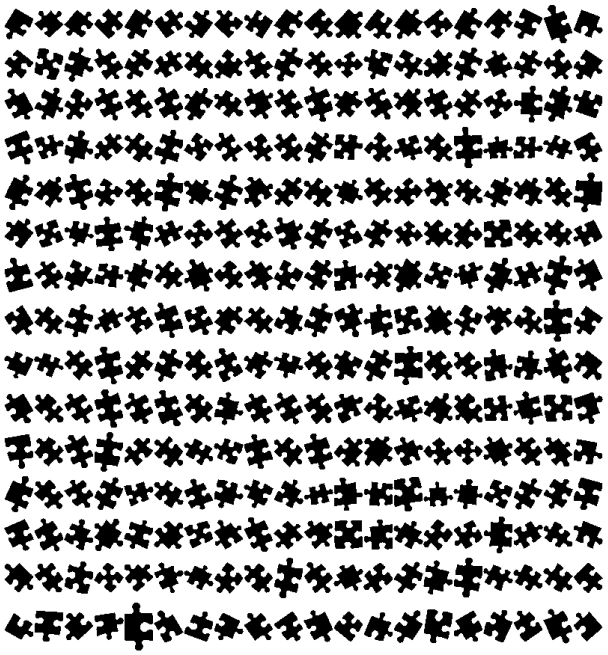

Fig. 1. Puzzle 5: the outlines of all the puzzle pieces

2. CONSTRUCTING THE TOPOLOGICAL SOLUTION

2.1. Classification of the puzzle pieces

Before we start with the actual construction of the topological solution, we classify the puzzle pieces. The classes are defined ac-

0-7803-8554-3/04/$20.00 ©2004 IEEE.

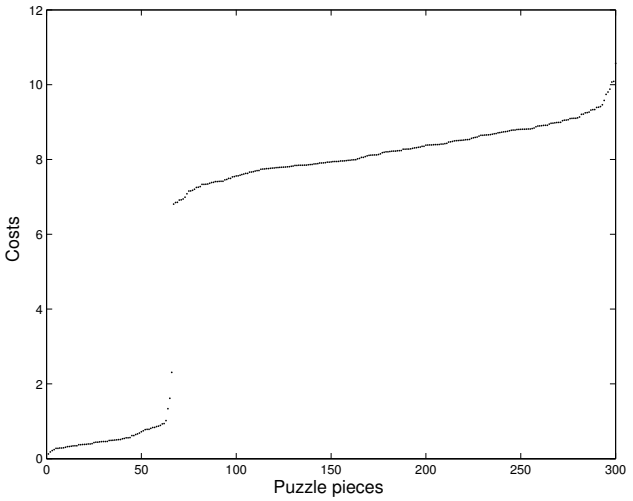

Fig. 2. Sorted costs after matching with an artificial straight edge

cording to the number of straight edges of each piece:

- **corner edge piece**: two straight edges, the four corners of the jigsaw puzzle.
- **one edge piece**: one straight edge, together with the corner edge pieces they form the rectangular frame of the jigsaw puzzle.
- **internal piece**: no straight edge, the pieces inside the rectangular frame.

To implement this classification, we reuse the developed contour matching algorithm. We first match all pieces with an artificial right corner. The pieces that produce the four lowest matching costs, most strongly resemble an artificial right corner and should correspond to the corner edge pieces we are looking for. Next, we repeat the matching procedure but with an artificial straight edge. The pieces that produce the lowest costs should resemble an artificial straight edge. After sorting these costs, we obtain a clear separation between edge pieces and internal pieces; see figure 2. Using the information gathered from these two procedures, we can perfectly classify all the pieces in the three described classes.

2.2. General solving framework

We first define what we mean with the term topological solution. A topological solution consists of a unique position for each puzzle piece in a rectangular grid. During the scanning procedure we kept track of the correct row and column numbers for each piece in the manually laid jigsaw puzzle. After finding a topological solution, we thus have a unique row and column couple for each position in the grid. We now can check the correctness of a topological solution by verifying the couples in the grid. An example of a correct topological solution is given in figure 3; mirrored and rotated versions of this correct topological solution are also considered correct.

The global goal is now to devise an algorithm that gives as output the topological solution that has the highest probability to be a correct topological solution. If we want to make this algorithm as simple as possible, it is necessary to split the problem into

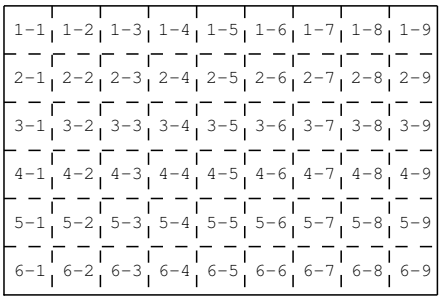

Fig. 3. Example of a correct topological solution

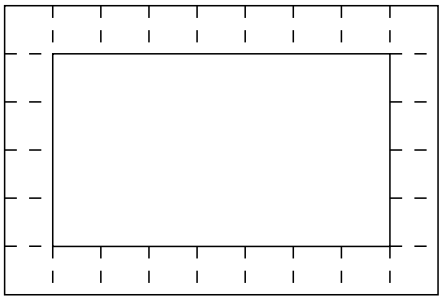

Fig. 4. Possible positions in a rectangular grid for the edge pieces

subproblems. We achieve this by first trying to find the correct topological solution for the edge pieces: the edge topology. The possible positions in a rectangular grid for the edge pieces are displayed in figure 4, we will call this layout the rectangular frame. The classification step has already delivered the necessary perfect split between edge pieces and internal pieces. The reduction of the logical complexity also results in a reduction in search space, i.e. we can completely ignore the internal pieces.

Every position inside the rectangular frame has two neighbor positions within the rectangular frame. We can now define two requirements for a valid edge topology: the edge topology must form one cycle containing all edge pieces, and edge pieces that are neighbors must share a common contour and thus must have a low matching cost. The solving algorithms for the edge topology are based on these requirements. After finding the edge topology we can finally determine the correct number of rows and columns for the rectangular grid by checking the positions of the four corner edge pieces within the edge topology. We then fit the cycle of edge pieces in the rectangular frame with the edge corner pieces onto the corner positions.

Now we will use the already determined edge topology as the starting point for finding the correct topological solution for the internal pieces: the internal topology. Before each iteration of the solving algorithm for the internal topology, we determine the "trusted positions". With a trusted position we mean a position in the rectangular grid that has two or more already fixed neighbor pieces (called eligible pocket in [5]). A few examples are given in figure 5. They are called trusted positions because they have enough neighbors with which we can calculate the matching cost. The solving algorithms for the internal topology are also based on the requirement of the pieces having a low matching cost with all

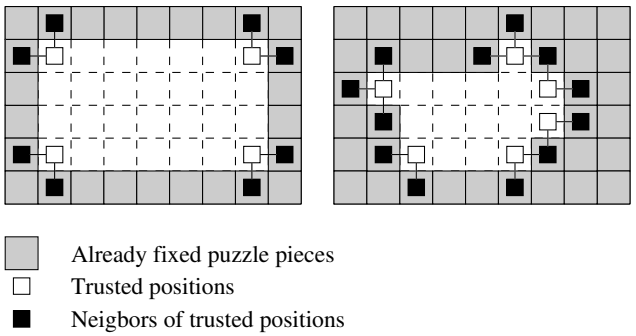

Fig. 5. Examples of trusted position

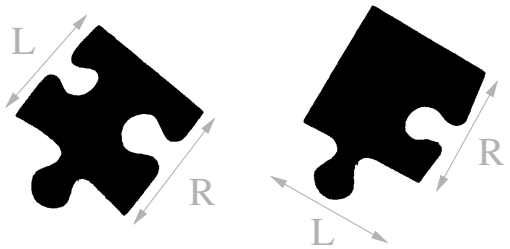

Fig. 6. The two sides of the edge pieces used for matching

neighbors. The algorithms will use the trusted positions as input and will give as output one of the trusted positions together with one of the still to be fixed internal pieces. This piece will then be fixed in that position. If we repeat the procedure for all the remaining internal pieces, we can completely fill the rectangular grid.

2.3. Puzzle piece treated as one contour

In this section we describe the solving algorithms that treat a puzzle piece as one contour. We thus obtain one matching cost per pair of pieces.

2.3.1. Solving algorithm for the edge topology

We first match all edge pieces with each other. Conceptually, we can describe this by a graph: a node represents a piece and an edge represents the match between a pair of pieces and has the matching cost as attribute. Following the requirements we have to find a cycle in the graph that visits all nodes once. The costs of the edges of the cycle must be as low as possible. This can be treated as a Symmetric Traveling Salesman Problem where we look for the cycle that has the lowest total cost (sum of all the costs). But this method does not necessarily give the edge topology that has the highest probability to be the correct one, because it treats all edge costs in the same way. A better way of finding the cycle is using a highest confidence first algorithm. This highly favors the most confident matches or the matches with the highest probability to be correct. For each iteration of the algorithm we do the following things to select the next edge of the cycle. We first search for all free nodes k. Free nodes are nodes from whom are selected less than two edges for the cycle. For each node k we then create a list of edges of that node: L_k. We delete all the edges from the lists L_k that are already selected for the cycle, that have as second node a non-free node or that create a subcycle. Next, we search in each list L_k the edge a_k with the lowest cost $c(a_k)$ and the edge b_k with the second lowest cost $c(b_k)$ and calculate $c(b_k) - c(a_k)$. Finally the edge a_k corresponding with the highest $c(b_k) - c(a_k)$ (evaluated over all L_k) is selected as the next edge for the cycle. If we keep iterating the edge selector, we obtain the complete cycle and thus the edge topology.

2.3.2. Solving algorithm for the internal topology

Here, we also use a highest confidence first algorithm (as described in [5] for this context). For each iteration of the algorithm we do the following things to select one trusted position and one still to be fixed internal piece. We use all the trusted positions k as input. For each position k we create a list of costs L_k. We obtain L_k by calculating for each still to be fixed piece i, the cost of fitting it in position k. This fitting cost is the mean of the matching cost of piece i with all neighbors of position k. Next, we search for each list L_k the piece a_k with the lowest cost $c(a_k)$ and the piece b_k with the second lowest cost $c(b_k)$ and calculate $c(b_k) - c(a_k)$. Finally the trusted position with the highest $c(b_k) - c(a_k)$ (evaluated over all L_k) is selected together with internal piece a_k.

2.4. Puzzle piece treated as four linked subcontours

In this section we describe the solving algorithms that treat a puzzle piece as four linked subcontours, corresponding with the four sides of a piece. We thus obtain one matching cost per pair of subcontours. To locate the four subcontours in the shape vector we locate the four corner points of the piece that delimit the four sides of a piece. We can locate the corner points by applying some checks on the shape vector. The detailed explanation is given in [10].

2.4.1. Solving algorithm for the edge topology

Here we also first match all edge pieces with each other, but now we can use additional information: the four sides of the pieces. We can limit the algorithm to two sides for each edge piece. These sides are displayed in figure 6 for the two types of edge pieces. Using the labels like they are displayed, side L will always fit with side R in the edge topology. We thus match side L with side R and side R with side L respectively for each pair of edge pieces. With this method we obtain two matching costs per pair of pieces. Here we also can describe the matches with their costs by a graph, but now a directed graph. We now have to find a cycle in the graph that visits all nodes once and respects the directional restrictions of the edges. The costs of the edges of the cycle must again be as low as possible. This can be treated as an Asymmetric Traveling Salesman Problem (as described in [5] for this context). But for the same reason we reuse the highest confidence first algorithm used for the undirected graph. Of course we adapt it so it takes the directional restrictions into account.

2.4.2. Solving algorithm for the internal topology

Here we can also use the additional information to make the solving algorithm much more robust. The algorithm is the same as the version for one contour, except for the calculation of the cost of fitting an internal piece in a trusted position. By splitting the pieces in four sides, we now can keep track of the orientation of

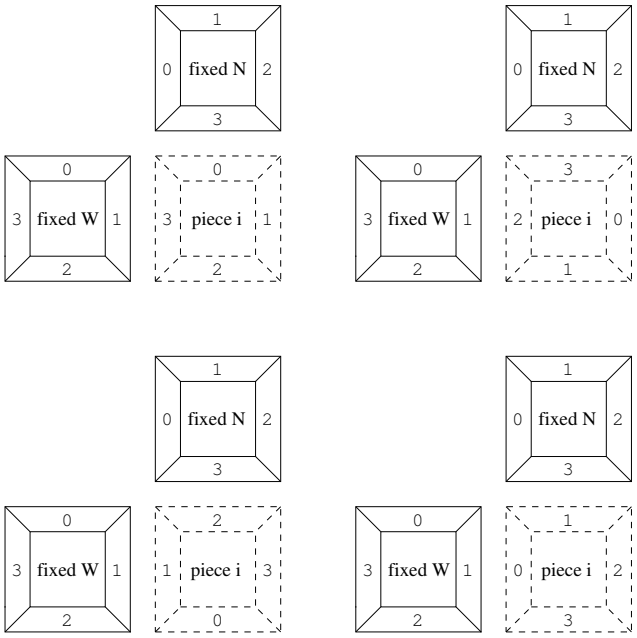

Fig. 7. Calculation of the fitting cost using the sides information

each already fixed piece. The method we use is displayed in figure 7 for a trusted position with two already fixed neighbors. We displayed the pieces conceptually by a square with four labeled sides. For the four possible rotations of piece i, we calculate the mean of the matching costs between each pair of subcontours that has to fit. For example, for the first orientation in figure 7 this is the mean of (subcontour 3 of fixed N, subcontour 0 of piece i) and (subcontour 1 of fixed W, subcontour 3 of piece i). From the four costs, the lowest one is selected as the final fitting cost. We store the orientation that gives that lowest cost, it will become the final orientation of the piece when it is selected as the to be fixed piece. The calculations on the lists L_k remain the same.

3. RESULTS AND CONCLUSION

We tested the described algorithms for the construction of the topological solution on five different jigsaw puzzles. They differ in size of the pieces, number of pieces, manufacturer, diversity in tab configuration and typical contour outline. For each jigsaw puzzle we increased the accuracy of the matching algorithm until the solving algorithms could construct a correct topological solution. The total running time (scanning and contour extraction excluded) was then measured and is given in table 1. The algorithms were implemented in C, compiled with gcc 3.2.3 and run on a AMD Athlon XP 1700+. Generally the running time increases with the amount of pieces and with the required accuracy. The required accuracy decreases with the diversity of the contour outlines. This explains why the running time is so long for puzzle 3 compared to puzzle 4 (for the one contour version): the diversity in the contour outlines was much less than in puzzle 4, so we had to increase the accuracy to still obtain a correct topological solution. The four subcontours version is less dependent on the diversity because it uses more a priori information. This explains why we could find a correct topological solution for puzzle 5 with the four subcontours version, but not with the one contour version. This also explains a normal time for puzzle 3 for the four subcontours version. Ultimately, we were able to solve a jigsaw puzzle consisting of 300 puzzle pieces, the largest one solved automatically to date, in an extremely fast time.

	running times (in seconds)	
	one contour	four subcontours
Puzzle 1: 4x7	4	6
Puzzle 2: 6x9	33	10
Puzzle 3: 13x8	153	27
Puzzle 4: 12x9	81	29
Puzzle 5: 15x20	-	244

Table 1. The running times for the tested jigsaw puzzles

4. REFERENCES

[1] T. Altman, "Solving the jigsaw puzzle problem in linear time," *Applied Artificial Intelligence*, vol. 3, no. 4, pp. 453–462, 1989.

[2] R. W. Webster, P. S. LaFollette, and R. L. Stafford, "Isthmus critical points for solving jigsaw puzzles in computer vision," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 21, no. 5, pp. 1271–1278, 1991.

[3] H. Bunke and G. Kaufmann, "Jigsaw puzzle solving using approximate string matching and best-first search," in *Computer Analysis of Images and Patterns*, D. Chetverikov and W. G. Kropatsch, Eds. 1993, vol. 719 of *Lecture Notes in Computer Science*, pp. 299–308, Springer-Verlag.

[4] M. G. Chung, M. Fleck, and D. A. Forsyth, "Jigsaw puzzle solver using shape and color," in *Proc. of the 4th International Conference on Signal Processing*, 1998, pp. 877–880.

[5] D. Goldberg, C. Malon, and M. Bern, "A global approach to automatic solution of jigsaw puzzles," in *Proc. of the 18th annual symposium on Computational geometry*, 2002, pp. 82–87.

[6] F. H. Yao and G. F. Shao, "A shape and image merging technique to solve jigsaw puzzles," *Pattern Recognition Letters*, vol. 24, no. 12, pp. 1819–1835, 2003.

[7] P. De Smet, J. De Bock, and E. Corluy, "Computer vision techniques for semi-automatic reconstruction of ripped-up documents," in *SPIE AeroSense Proc. 5108B, Investigative Image Processing 3*, 2003.

[8] P. De Smet, J. De Bock, and E. Corluy, "Semi-automatic jigsaw puzzle reconstruction of fragmented documents," in *3rd Triennial Meeting of the European Academy of Forensic Science*, 2003.

[9] P. De Smet and E. Corluy, "High-precision recomposition of fragmented 2-d objects," in *14th ProRISC workshop on Circuits, Systems and Signal Processing*, 2003.

[10] J. De Bock, "Computer algorithms for solving jigsaw puzzles," M.S. thesis, Ghent University, 2003.