Applied Artificial Intelligence: An International Journal
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/uaai20

SOLVING THE JIGSAW PUZZLE PROBLEM IN LINEAR TIME

TOM ALTMAN [a]
[a] Department of Computer Science , University of Kentucky , Lexington, Kentucky, 40506
Published online: 27 Apr 2007.

SOLVING THE JIGSAW PUZZLE PROBLEM IN LINEAR TIME

TOM ALTMAN
Department of Computer Science, University of Kentucky, Lexington, Kentucky 40506

We introduce an algorithm that efficiently matches (fits together) parts of boundaries of two-dimensional objects in order to assemble apictorial jigsaw puzzles. A rotation-independent shape encoding allows us to find the best (longest) match between two shapes in time proportional to the sum of the lengths of their representations. In order to find this match, we use Weiner's string matching technique combined with compact position trees to find, in linear time, the longest shared pattern between two strings. The shape matching procedure is then used by two greedy algorithms to assemble the apictorial jigsaw puzzles.

INTRODUCTION

We introduce an algorithm that efficiently matches (fits together) parts of boundaries of two-dimensional objects in order to assemble apictorial jigsaw puzzles. A rotation-independent shape encoding (Altman and Easwar, 1987) allows us to construct a procedure that finds the longest match between two shapes in time proportional to the sum of the lengths of their representations. To find this match, we use Weiner's string matching technique (Weiner, 1973) combined with compact position trees to find, in linear time, the longest shared pattern between two strings. The shape matching procedure is then used by two greedy algorithms, which use the *fuse longest match first* approach, to assemble the jigsaw puzzles.

Background

A natural testing ground for techniques of boundary matching is found in the *jigsaw puzzle problem* (JPP): given a set of simply connected planar regions (puzzle pieces) whose only significant features are their boundaries, is it possible to rotate and translate each piece so that the pieces fit together into some (e.g., rectangular) region with no holes or overlaps between them (Pavlidis, 1978; Radack and Badler, 1982). We assume that any two adjacent pieces of the JPP share at least three consecutive points along their boundaries. An example of a random apictorial jigsaw puzzle, used in the testing of our algorithms, is shown in Fig. 1. Although the JPP is usually used as a tool to evaluate the performance of shape matching algorithms, the problem of puzzle assembly is,

Key words: Jigsaw puzzles, shape encoding, shape matching.

Applied Artificial Intelligence: 3:453-462, 1989

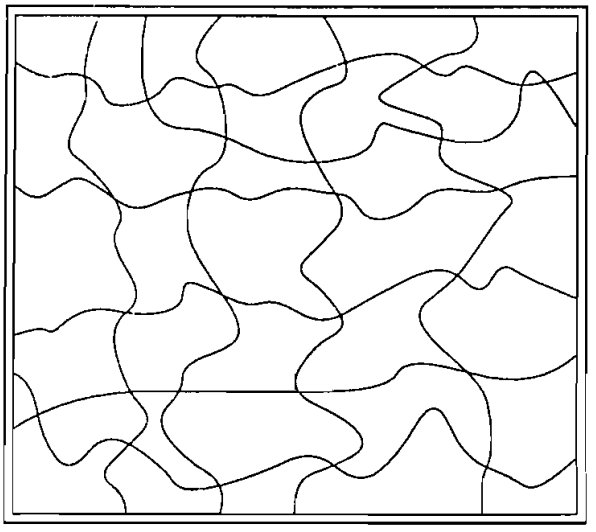


FIGURE 1. A randomly generated jigsaw puzzle.

by itself, an interesting one from the computational complexity point of view. As we show in the next section, even restricted versions of the JPP are NP-complete. We then describe the rotation-invariant boundary encoding that will be used by the linear time shape matching algorithm to fuse two compatible pieces into one. Finally, the puzzle assembly algorithms are presented.

COMPLEXITY OF THE JPP

Besides the shape matching aspects of this problem, we are also interested in the development of assembly algorithms for the JPP. As it turns out, the JPP is an intractable problem even with the help of a *perfect* shape matching oracle. For the time being, let us restrict our domain of problems to a special class, JPP′, which is formally defined as follows:

Instance. A rectangular region B, whose corners are located at the coordinates $\{(0,0), (1,0), (1,4), (0,4)\}$. A set of n rectangular jigsaw pieces, each of height 2 and a rational width $s_i < 1$. The pieces are described by quadruples of the form $\{(0,0), (s_i,0), (s_i,2), (0,2)\}$; also

$$\sum_{i=1}^{n} s_i = 2$$

(i.e., the sum of the areas of the n pieces is 4, the same as that of B, shown in Fig. 2).

Question. Is it possible to map each quadruple $\{(0,0), (s_i,0), (s_i,2), (0,2)\}$ to $\{(0 + w_i, h_i), (s_i + w_i, h_i), (0 + w_i, 2 + h_i), (s_i + w_i, 2 + h_i)\}$, where $h_i \in \{0,2\}$ and $0 \leq w_i < 1$, so that they all fit within B without holes and/or overlaps? In other words, for each piece we must decide if it is to go in the upper or lower region of B.

Theorem 1. The JPP′ is NP-complete.

Proof. Clearly JPP′ is in NP. For each piece, we can choose (nondeterministically) whether it belongs to the upper ($h_i = 2$) or lower ($h_i = 0$) portion of the rectangle B. The computation of the horizontal offset w_i is straightforward. In the (deterministic) verification of the answer, the testing for overlaps

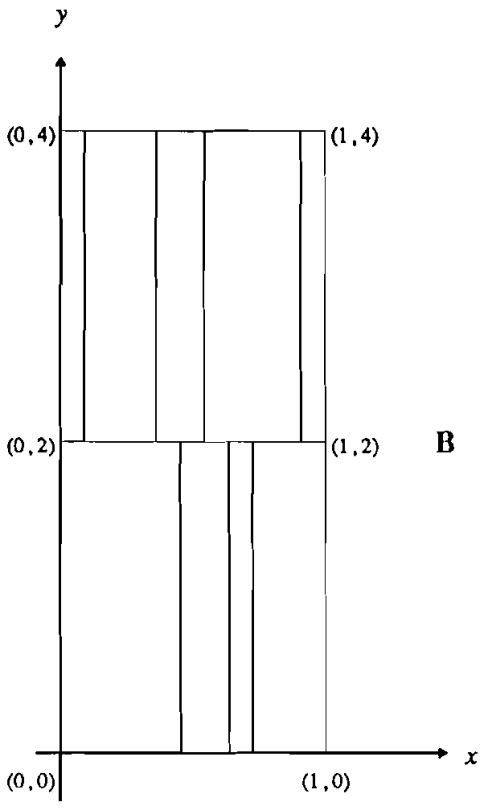


FIGURE 2. Example of an NP-complete JPP.

between any two pieces takes constant time and the procedure can be repeated $n^2/2$ times to check for any possible overlaps. Since the absence of overlaps also implies absence of holes, we need not check for the latter.

The *set partition problem* (SPP) is reducible to JPP′. As described in Garey and Johnson (1979, pp. 223, SP12), SPP requires one to determine whether a set of weighted elements can be partitioned into two disjoint subsets of equal weight. It is obvious that the SPP is a special case of the JPP′; hence, by restriction, JPP′ is NP-complete.* □

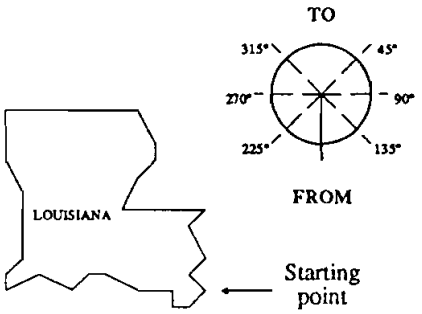
Despite the discouraging consequences of Theorem 1, we will present two polynomial time jigsaw puzzle assembly algorithms for a particular class of JPP. Our problem domain will consist only of puzzles whose pieces are sufficiently unique that they can be assembled without backtracking. This requirement is not too restrictive since most real jigsaw puzzles have this property, called *unique interlock* (Pavlidis, 1978). Observe that the JPP′ puzzles do not possess this property.

THE ENCODING METHOD USED

A number of methods have been used to encode boundaries of two-dimensional regions. For an excellent survey on the topic see, e.g., Pavlidis (1978) and Radack and Badler (1982). Freeman and Garder (1964) were the first to introduce a method of *chain coding* of a boundary. There, a curve (shape) is represented by a beginning point followed by a sequence of lines, each starting at the endpoint of the previous one. The directions and lengths of the lines are restricted so that they can be represented with a relatively small number of symbols. The major problem with the *Freeman chain code* is that it is not rotation invariant [i.e., rotation will drastically affect the coding of a curve; see Radack and Badler (1982) for specific examples].

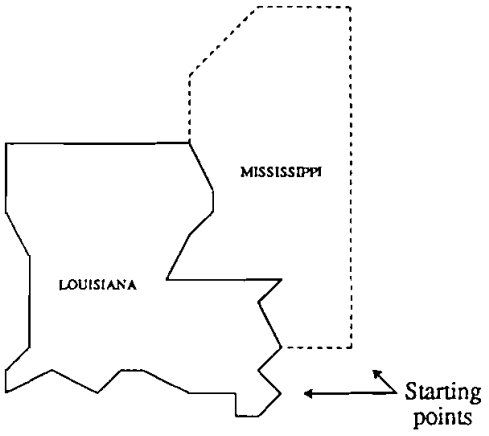
The chain coding method used here falls in the general class of external space domain methods and is based on a scheme presented in Altman and Easwar (1987). It is rotation invariant; hence, the encoding of a shape is independent of its placement on the xy plane. Whereas the Freeman chain code uses *absolute* directions of line segments, usually of uniform length, we represent their *relative* directions (i.e., the changes in directions between respective line segments) and with a variable length. For a given shape S_i, which is described by a sequence of xy coordinates of k boundary points, the (α,d)-*encoding* is defined to be a sequence $\{\alpha_1, d_1, \alpha_2, d_2, \ldots, \alpha_k, d_k\}$ of (angular change, distance) pairs, as shown in Fig. 3. Each α_i, encoded as an integer between 1 and 359 degrees, represents the angular change between the line segments de-

*By reducing the two-dimensional bin packing problem to a more general form of JPP, we could also show that the jigsaw assembly problem is NP-complete in the strong sense.



The counterclockwise (α, d)-encoding for LOUISIANA is:

270,2,90,2,300,5, **60,2,225,25,120,5,15,2,315,1,330,5,** 300,64,270,9,330,5,
30,16,45,2,315,1,240,5,60,5,285,2,45,1,30,5,330,2,90,1,270,1,315,2



The clockwise $(360-\alpha, d)$-encoding for MISSISSIPPI is:

90,9,60,5, **60,2,225,25,120,5,15,2,315,1,330,5,** 30,9,45,18,45,16,90,225

The counterclockwise (α, d)-encoding for the fused shape is:

270,2,90,2,45,9,270,225,270,16,315,18,315,9,90,64,270,9,330,5, ... (unchanged)

FIGURE 3. The matching of two shapes.

fined by the points $(i - 1, i)$ and $(i, i, + 1)$. Each d_i is the square of the distance between the points i and $i + 1$. The distance is squared to avoid the problem of truncation in the representation of irrational numbers, e.g., $\sqrt{2}$, $\sqrt{5}$. It is clear that this encoding scheme is rotation invariant since the directional changes are themselves independent of the angle from which a given shape is examined.

THE SHAPE MATCHING ALGORITHM

The problem of matching two shapes may be stated as follows: given two (α,d)-encodings for shapes S_i and S_j, find the longest (in number of points) match between them. Observe that if two curves match, then the clockwise representation of the first (α,d)-encoding will be identical to the counterclockwise $(360 - \alpha,d)$-encoding of the second; e.g., see Fig. 3. Hence, to locate the longest match between two shapes, we need to find the longest common substring shared by the clockwise and counterclockwise encodings representing the shapes in question.*

The above is a classical problem in pattern matching (Aho et al., 1974). Using the techniques described in Weiner (1973), we construct a *position tree* for the string $z = xx : yy \$$ where x, y are strings corresponding to the (α,d)-encodings of the two shapes to be matched and : and \$ are symbols not in the alphabet used for the encoding. The position tree is then used to identify the longest repeated substring of z by looking for an interior vertex of the tree with the greatest depth. Additional care is taken to avoid longest repeated substrings of the form xx or yy. Algorithms for finding the longest common substring are described in detail in Aho et al. (1974) and Weiner (1973). Note that, to ensure the linearity of the shape matching procedure, we must use the *compact* version of the position tree given in Weiner (1973).

The joining of the two shapes along their longest match is now trivial: knowing the positions in the respective (α,d)-encodings where the longest match took place, we eliminate the shared substrings, recode the clockwise string to a counterclockwise representation, and concatenate the resulting strings into one. The codes of the line segments at the two connection points are adjusted accordingly.

Detection of Overlaps

As described, the above shape matching algorithm is unable to detect overlaps between the fused shapes occurring *outside* the region of the longest match. Figure 4 shows two possible types of such overlaps. Testing for the *primary*

*Since the starting points of the encodings are random and the longest match may occur *around* the starting points, we duplicate the encoding for each shape to ensure a match that is truly the longest one.

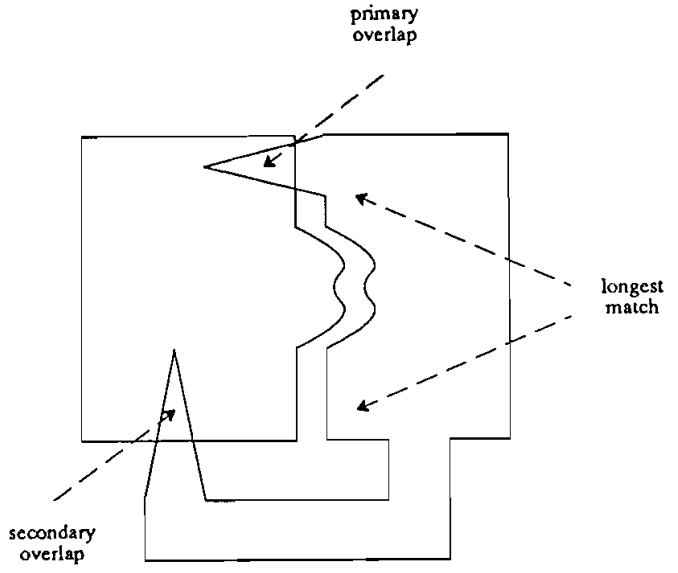


FIGURE 4. Primary and secondary overlaps.

Figure 4 shows two possible types of such overlaps. Testing for the *primary* (i.e., immediate) overlaps is a straightforward procedure that can be performed in constant time.

On the other hand, the *secondary* overlaps do pose an additional problem, since in the worst case we must check for possible *intersections* between the two shapes up to a distance of $O(k)$ away from the point of fusion, where k is the number of points describing the smaller shape. Although this may be accomplished in linear time (see, e.g., Guibas and Seidel, 1986), the problem of finding the *longest* match *without* overlaps still does not appear doable in linear time because of the possibility of *multiple* matchings between shapes, as shown in Fig. 5. There, the longest (as well as the second, third, . . . , qth longest) match may still be one with a primary and/or secondary overlap. Note, however,

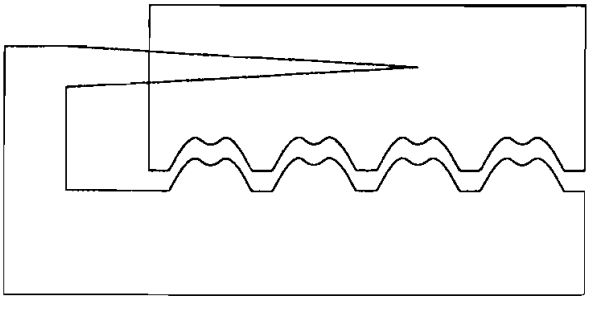


FIGURE 5. Multiple matchings between two shapes.

that the unique interlock restriction also eliminates the possibility of multiple matchings between any two shapes. Hence, there is no need to check for secondary overlaps by the shape matching procedure.

THE JIGSAW ASSEMBLY ALGORITHMS

In this section we present two jigsaw puzzle assembly algorithms, both of which use the shape matching algorithm of the preceding section. The first algorithm runs in time proportional to I, the number of points used to describe a given instance of a JPP. The running time of the second algorithm is $O(mI)$, where m is the number of pieces in the puzzle.

The jigsaw puzzle assembly algorithm that runs in time linear in I consists of two phases. For a given shape S_i, let γ_i *and* δ_i represent the counterclockwise and $(360 - \alpha_i)$-clockwise (α,d)-encodings, respectively.

In the first phase, we construct a string z, which has the following form:

$$z = \gamma_1\gamma_1\#\gamma_2\gamma_2\#. \; . \; .\#\gamma_m\gamma_m : \delta_1\delta_1\#\delta_2\delta_2\#. \; . \; .\#\delta_m\delta_m \; \$$$

Next, we construct a compact position tree for z. The number of nodes in this tree is proportional to the total number of points used to encode a given JPP. The construction of this tree is also performed in time linear in the total number of points (Weiner, 1973).

In the second phase, the deepest internal nodes of this tree, which indicate the longest matches between shapes, are examined. The leaf names, in the subtree whose root is the internal node currently being inspected, identify the two matching pieces in the puzzle. The algorithm continues until no more possible matches can be established. At that point, the unique interlock property guarantees that the puzzle is assembled properly.

In practice, however, this linear time algorithm did not perform as well as expected due to the sizes of the position trees constructed in the first phase.

The second algorithm is a straightforward application of the longest match procedure:

```
Compute the (α,d)-encodings for the m puzzle pieces;
For i: = 1 to m - 1.
  Find L, the piece with the longest (α,d)-encoding;
  Find the best match between L and the remaining m - i pieces;
  Fuse L and the piece with the best (longest) match;
  Adjust the (α,d)-code of the ''new'' L
End for.
```

The running time of this algorithm is clearly $O(mI)$, where I is the total number of points used to encode the m pieces of the jigsaw puzzle.

TABLE 1. Length of Strings vs. Time to Find Longest Match

Length of strings		
1	2	Execution time (s)
29	27	0.004
51	43	0.017
76	55	0.025
103	92	0.029
156	101	0.034
255	211	0.045
337	293	0.051

CONCLUSION

We have presented two jigsaw puzzle assembly algorithms that use a linear time shape matching procedure to find the longest match between shapes. The two algorithms were implemented on a Vax-11/750 and tested on a number of randomly generated puzzles. Whereas the first algorithm proved to be impractical for even medium-sized puzzles, the second one assembled relatively large jigsaw puzzles in a reasonable amount of time, as shown in Tables 1 and 2.

More detailed testing and further enhancements of the assembly algorithm, such as the inclusion of additional attributes (e.g., *color* and *texture*) into the shape matching procedure, are in progress. While the modifications are not straightforward, the linearity of the shape matching procedure will still be preserved, although some restrictions on these new puzzles will have to be imposed. For example, the color and/or texture in the JPP should not be allowed to change *along* the cut-lines between any two pieces; i.e., the shape cut-lines cannot overlap with the color or texture cuts.

TABLE 2. Number of Pieces vs. Puzzle Assembly Time

Number of pieces[a]	Time required for assembly (s)
6	0.6
8	0.9
10	1.5
12	2.1
14	2.7
16	4.0
18	6.1
20	7.9
22	10.8
24	13.5

[a]The average number of points per piece is 35.

REFERENCES

Aho, a. V., Hopcroft, J. E., and Ullman, J. D. 1974. *The Design and analysis of Computer Algorithms.* Reading, Mass: Addison-Wesley.

Altman, T., and Easwar, S. 1987. Rotation-Invariant Encodings for Linear-Time Shape-Recognition Algorithms. Tech. Report 89-87, Univ. of Kentucky, Lexington.

Freeman, H., and Garder, L. 1964. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Trans Electron Comput* EC-13(2):118–127.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: W. H. Freeman.

Guibas, L. J., and Seidel, R. 1986. Computing convolutions by reciprocal search. *Proc. Second Annual Symposium on Computational Geometry,* pp. 90–99. June 2–4, Yorktown Heights, NY.

Pavlidis, T. 1978. A Review of Algorithms for Shape Analysis, *Comput Graphics Image Process* 7:243–258.

Radack, G. M., and Badler, N. I. 1982. Jigsaw puzzle matching using a boundary-centered polar encoding. *Comput Graphics Image Process* 19:1–17.

Weiner, P. 1973. Linear pattern matching algorithms. *Conf. Record, IEEE 14th Annual Symposium on Switching and Automata Theory,* pp. 1–11. October 15–17, Iowa City, Iowa.