# Reassembling Shredded Document Stripes Using Word-Path Metric and Greedy Composition Optimal Matching Solver

Yongqing Liang and Xin Li, *Senior Member, IEEE*

***Abstract*—This paper develops a shredded document reassembly algorithm based on character/word detection. A new word compatibility estimation metric and a searching strategy called Greedy Composition and Optimal Matching (GCOM) are proposed to compose documents from their vertically shredded stripes. We reduce the stripe puzzle reassembly problem to the traveling salesman problem (TSP) on a sparse graph. The word-path compatibility metric takes advantages of the optical character recognition (OCR) to compute the compatibility score among a group of stripes. The global composition strategy, based on an integration of greedy composition and optimal matching, is proposed to search for a maximal Hamiltonian path and the final global reassembly. We demonstrate that our solver outperforms the state-of-the-art puzzle solvers on reassembling stripe shredded documents.**

***Index Terms*—Shredded document reassembly, sequence compatibility measurement, global reconstruction from local alignments.**

## I. Introduction

REASSEMBLING the remnants of destroyed (shredded or ripped) documents enables recovery of valuable information in forensic investigations and archival research. In the field of information security, reassembling document could also help us understand our limitations against adversaries' attempts to gain access to information. However, so far, such restoration remains as a daunting task. During the 2011 DARPA Shredder Challenge [1], about 9000 teams participated in a competition to develop programs that piece together 9 shredded pages. The winning solver was a semi-automatic system which finds inter-fragment matching using boundary contour and strokes. Then, with the help of tremendous user interaction (which eventually took 3 weeks, more than 600 man-hours), the winning solver semi-manually explored different possible combinations, and managed to reconstruct the majority of 3 pages and part of the others. As a result, due to the enormous amount of manual intervention needed, researchers commonly believe that such reconstruction from shredded documents is prohibitively difficult. A computerized system capable of automatically restoring fragmented archives into their complete digital form is not only a significant time saver, but also a valuable tool to help convert these physical ruins to computer-understandable digital representations for subsequent machine-assisted analysis.

Here, we aim to first explore a simpler variant of this problem, namely, *the reassembly of shredded document stripes*, where the document is vertically cut, then the pieces are rectangular and already oriented. We refer to this problem as the reassembly of *document stripe puzzle*, which mimics the output from stripe-cut shredding machines. Effectively solving this simpler problem will help pave the way for resolving the original more general shred reconstruction (where pieces are cut by cross-cut shredding machines).

The solution for this document stripe puzzle is to find an optimal permutation that correctly re-orders the given stripes so they recompose the original image. Effectively solving this document stripe reassembly is closely related to the globally consistent searching the set of image poses from unreliable local alignments, image feature modeling, and partial matching/alignment computation. Algorithms developed to tackle these issues also have broad applications in tasks such as wide-baseline image panorama, image-based reconstruction, structure-from-motion, etc. However, even reliably solving image stripe reassembly is difficult, since the puzzle solver needs to effectively tackle challenging technical issues in both *local matching* and *global composition*.

Effective *local matching* to align each stripe to its correct neighbors is a critical first step in puzzle solving. Suitable features and metrics need to be developed to evaluate how likely two pieces are adjacent pairs in the original image. Pixel-based features and metrics are widely studied in existing literatures [2]–[5]. They evaluate the matching compatibility between adjacent pieces using similarity of pixel colors or their gradients along the abutting boundary. These metrics assume the original image contents are smooth. This assumption is generally reasonable for common image puzzles. However, if

Manuscript received February 21, 2019; revised July 8, 2019 and August 27, 2019; accepted August 29, 2019. Date of publication September 18, 2019; date of current version April 23, 2020. This work was supported in part by the National Science Foundation under Grant IIS-1320959 and in part by the National Natural Science Foundation of China under Grant 61728206. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Engin Erzin. *(Corresponding author: Xin Li.)*

The authors are with the School of Electrical Engineering Computer Science, Louisiana State University, Baton Rouge, LA 70808 USA (e-mail: yqliang@cct.lsu.edu; xinli@cct.lsu.edu).

This paper has supplementary downloadable material available at https://github.com/xmlyqing00/DocReassembly, provided by the authors. The material includes source codes, evaluation dataset DocDataset, and a demo video.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2019.2941777

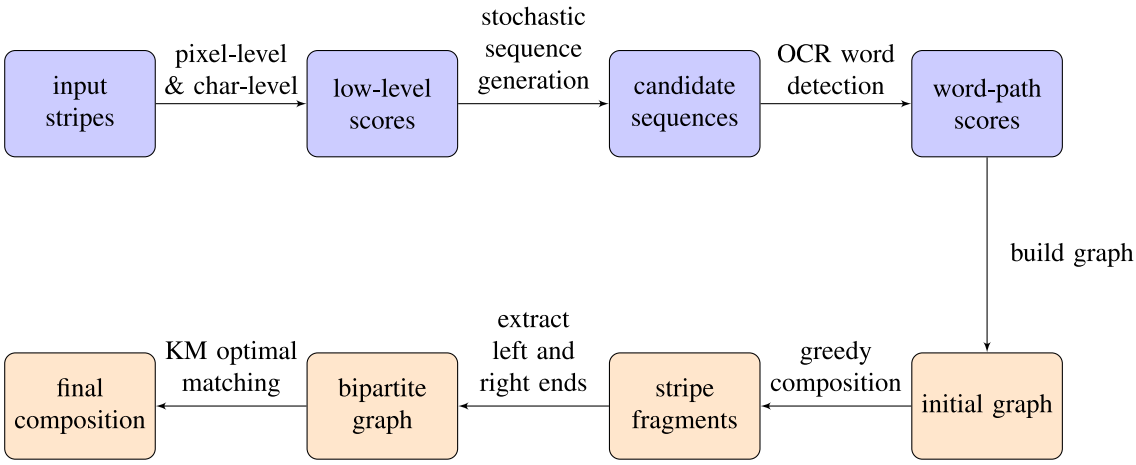



Fig. 1. A flowchart of the proposed algorithm, which consists of two parts: (1) local alignment (blue) and (2) global composition (orange). In the local local alignment stage: low-level metric helps group the input stripes into candidate sequences; then our OCR word detector measures compatibility scores among stripes and reduces this problem to a traveling salesman problem on a graph. In the global composition stage, we greedily compose stripes into fragments, then on a bipartite graph we compute the optimal matching using the KM algorithm.

the given puzzle is not an image but a text document, those printed or written characters are discrete, so colors and textures are often discontinuous. Thus, pixel-level smoothness metrics become less reliable. In this work, we explore more robust local matching using text semantics.

Because local matching is often inevitably ambiguous, a second *global composition* step to prune and optimize local matching sequences using global consensus is often needed towards robust fragment reassembly. Since this document stripe reassembly is a one-dimensional arrangement problem, if we treat each stripe as a vertex and the stitching between two stripes as an edge, we can model the solving of reassembly on a graph, which we refer to as a *reassembly graph* in the following. We propose to model this problem as finding a maximally weighted Hamiltonian path on this reassembly graph. Hence, this reduces the optimal composition to finding the optimal path in a traveling salesman problem (TSP). The TSP problem is formulated as follows: Given a set of vertices and a set of edges describing the weights between the vertices, to find the minimal (or maximal) weighted path that visits each vertices once. Directly enumerating all possible compositions is impossible because the search space is $O(N!)$, where $N$ stands for the number of the vertices in the composition graph, and here the number of stripes. General approximation TSP solvers, such as Christofides algorithm [6], gives bounded TSP tours (1.5 times the optimal path), but poses certain constraints (triangular inequality) on the edge weights which cannot be guaranteed in this problem setting. Other greedy strategies adopted in reassembly literatures, such as [4], [5], [7]–[13] iteratively grow the path by selecting locally optimal edges.[1] These strategies work better for small puzzles or when compatibility evaluation is reliable. But when puzzles become bigger and compatibility evaluation becomes unreliable, incorrect compatibility evaluation will mislead the greedy composition and traps the algorithm to local minima easily. In this work, we propose a new integrated greedy composition and optimal matching (GCOM) algorithm, to support more robust and efficient global searching in this problem.

Our **main idea** is based on the following two observations: (1) High-level semantic text information, such as characters or words, provides useful clues to measure matching compatibility. If a subsequence of stripes could compose one or many words, this composition order is likely to be correct. Therefore, for local pairwise matching, we develop a new *word-path metric* to evaluate matching compatibility. (2) Stripes with high compatibility scores can be merged into fragments first greedily. Then, matching between fragments/stripes with lower certainty can be solved through optimal searching using algorithms such as Kuhn-Munkres algorithms [14] to produce a more robust final result. We call this new algorithm the integrated greedy composition and optimal matching (GCOM) algorithm. Our overall algorithm is summarized in Fig. 1.

The **main contributions** of this paper include:

1) We model the reassembly of document stripe puzzle as finding an optimal Hamiltonian path in a graph, which reduces to the traveling salesman problem (TSP) in a sparse graph, and propose an effective solver to solve it. Our algorithm outperforms existing reassembly strategies by a big margin in recomposing document stripe.
2) We design a first text-semantic based matching compatibility metric to support the local alignment computation of document stripes. This provides us useful high-level semantic information in evaluating partial matching between non-overlap document images.
3) We propose an integrated greedy composition and optimal searching (GCOM) algorithm to find a global composition. This greatly improves the robustness of pruning unreliable local alignments. Such composition

[1]Their compositions are not formulated as finding the Halmiltonian path, but we can still adopt their greedy searching strategies here.

models have promising applications in many wide-baseline image matching and reconstruction tasks.

4) We release a new dataset of digital document stripe puzzles to the scientific community for comparative studies.

## II. Related Work

Teaching computers to solve scanned Jigsaw puzzles has been posted as a classic AI problem in image processing and computer vision literature since [15]. Semi-automatic approaches [16], [17] develop a computer-aided de-shredder system for experts to reconstruct documents manually. Existing automatic approaches on this problem can be generally classified into two categories: the solving of regularly shaped puzzles (where pieces are usually rectangle or square), and the solving of irregularly shaped puzzles (where pieces are arbitrarily cut). In this paper, we consider the reassembly of a document image from rectangular oriented stripes, which is a type of regularly shaped jigsaw puzzles. For such kind of puzzles, matching of fragments using their geometric contours do not provide much information.

Most recent automatic puzzle solving algorithms adopt a two-step local-to-global solving scheme. (1) First, locally identify potential pairwise alignments between fragments. (2) Second, prune and compose local alignments globally, and solve the final poses of all the pieces. In the following, we classify and discuss different puzzle solvers according to their local matching and global composition strategies respectively.

### A. Local Alignment Estimation

**Image based features:** Most existing image/geometry puzzle solvers directly use the pairwise matching scores, based on geometry fitness [18]–[25], or smoothness of the color/intensity or its gradients [3], [5], [12], [25]–[37] for this evaluation, and then adopt a user-defined threshold to select which alignments to keep or discard. These measures are built upon the assumption that image contents in the original image have smooth color transitions. While this is reasonable for images, on fragmented documents, the contents are characters that have sharp discontinuity along their contours. Pixel color transitions are then less reliable.

**Text extraction using OCR:** Optical Character Recognition (OCR) is the technique to convert images of text into machine-encoded characters. OCR has been widely studied in computer vision and natural language processing fields and is utilized in many real world applications. Well-known OCR systems include Google Tesseract [38], Abbyy FineReader [39], and OnlineOCR [40]. A typical OCR pipeline analyzes the layout of given image, segments out texts/words regions, then performs character recognition. Xing *et al.* [41] observe that Chinese characters often have fixed shapes. Such recognition can be used to extract text-based features from document fragments to guide their alignments. Paixao *et al.* [42] build a network to measure the compatibility between shredded stripes. So, they use the estimated sizes of characters to guide stripe document matching. However, their method does not take semantic information into considerations.

### B. Global Composition

The locally computed pairwise alignments between stripes can be modeled using a graph, in which vertices correspond to stripes and edges correspond to their pairwise alignments. Then, solving the optimal composition from these alignments reduces to finding an optimal Hamiltonian path that visits every vertex exactly once. The composition problem becomes the traveling salesman problem (TSP).

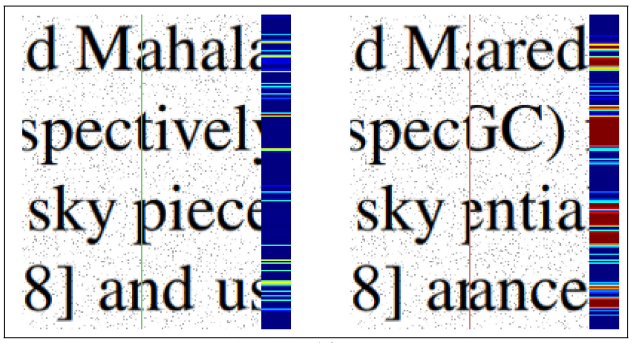
**General TSP solvers** can be classified into two types, direct solvers, and approximation solvers. Direct solvers enumerate all possible compositions to find the exact solution. But since the search space is of $O(N!)$ ($N$ being the number of vertices in the composition graph, namely, the number of stripes), their time complexity is prohibitive even when the fragment size is just moderate. Approximate methods have also been developed for TSP. For example, the Christofides algorithm [6] can suggest a TSP tour that is at most 1.5 times the optimal. However, many such algorithms require the weights of edges to satisfy certain criteria, such as triangular inequality, which cannot be guaranteed in our problem setting. Specifically, the triangular inequality constraint requires that for any three vertices $u, v, w \in V$, $c(u, w) \leq c(u, v) + c(v, w)$, where $c$ is the cost/weight function. However, here in the reassembly problem, the pairwise compatibility scores among stripes are independent and may not follow this inequality.

**Other related searching solvers** developed in fragment reassembly literatures include greedy growing algorithms, loop-based growing algorithms and genetic algorithm. Greedy growing algorithms [4], [7], [9]–[13], [19], [23], [29], [32], [34], [37] iteratively select edges by choosing the locally optimal pair. These strategies work better for small puzzles or when compatibility evaluation is reliable. But when puzzles become bigger and the compatibility evaluation becomes unreliable, incorrect compatibility evaluation will mislead the greedy composition and trap the algorithm to local minima easily. The loop-based growing algorithms [3], [30], [36], [43], [44] enforce mutual consistency among multiple pieces using loop closure constraints, and adopt hierarchical growing or back-tracking mechanisms to solve the global composition. However, loop-closure constraint does not fit here since our reassembly problem is one-dimensional. Genetic algorithms [5], [31], [35] utilize mutation operators of merging two "parent" solutions to an improved "child" solution. Nevertheless, genetic algorithms' search space are still too large to find the optimal solution.

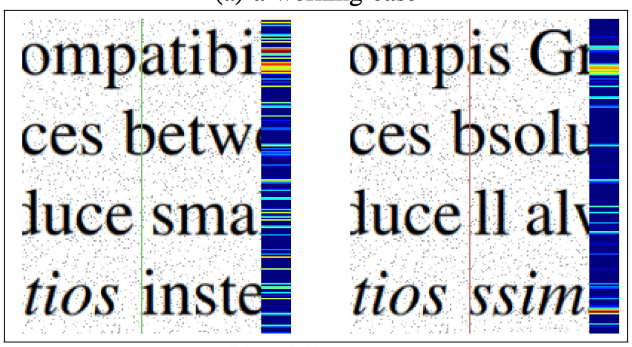
## III. Word-path Metric for Compatibility Estimation

The first step in puzzle solving is to identify potential local matches between stripe pairs. For document reassembly, we believe building an evaluator using higher-level semantic text information is beneficial. So we introduce a new text-based compatibility metric, named the *word-path metric*. It is a hierarchical compatibility evaluator containing two low-level metrics and a high-level metric.

Specifically, we measure pixel-level and character-level consistency in low-level metrics. They are both pairwise metrics and they only measure the compatibility along the abutting boundary



(a) a working case



(b) a failure case

Fig. 2. Examples of matching evaluation using pixel-level metric. The colorbar encodes dissimilarity along the abutting boundary in every row. Green and red seam lines indicate correct and incorrect alignments, respectively. A lower dissimilarity score indicates a better alignment in this metric. (a) shows a working case, where the dissimilarity score of a correctly aligned pair on the left is 12.1 and the incorrectly aligned pair has the score of 92.1. (b) shows a failure case: the dissimilarity score on correct and incorrect pairs are 23.1 and 18.6 respectively.

of two stripes. The low-level metrics are simple and fast, but they often introduce false positives (See Figs. 2 and 3). So, we also propose a higher-level metric using word information detected from two, or more often, multiple stripes. If a sequence of stripes composes meaningful words across their stitching boundaries, then such a composition sequence is likely to be correct. The pixel and character consistency provide rough pairwise compatibility scores that can help generate candidate stripe sequences for word detection.

### A. Low-Level Metric

*1) Pixel-Level Metric:* Pixel-level consistency is a natural way to measure the compatibility between two adjacent stripes. Large pixel color/intensity change indicates an unlikely matching. We can define the pixel-level compatibility score $C_p$ between a stripe pair $s_i$ and $s_j$ using a pixel-level dissimilarity score $d_p$. $d_p$ can be defined as the sum of absolute color differences of all neighboring pixels along the boundary,

$$d_p(s_i, s_j) = \frac{1}{H}\sum_{h=1}^{H} |s_i(h, W) - s_j(h, 1)|, \tag{1}$$

$$C_p(s_i, s_j) = \exp\left(-\max\left(0, \frac{d_p(s_i, s_j)}{\alpha_1} - \alpha_0\right)\right), \tag{2}$$

where $H$ and $W$ are the height and width of the stripe. $s_i(\cdot, W)$ denotes the color (intensity) of a pixel on the $W$-th column in stripe $s_i$. $\alpha_0$ and $\alpha_1$ scale the range of pixel-level compatibility score to [0,1]. In our experiments, we set $\alpha_0 = 1.5$ and $\alpha_1 = 20$. The higher value of $C_p$, the more alike this pair of stripes is. Note that $d_p(s_i, s_j)$ is always non-negative, and $C_p(s_i, s_j) \neq C_p(s_j, s_i)$, so the commutative law is not satisfied.

Figure 2 illustrates two examples of compatibility evaluation using pixel-level metric. The jet colormap shows the dissimilarity in every row. The green and red seam lines between two stripes indicate the correct and incorrect matching pairs, respectively. A lower dissimilarity score indicates a better alignment in this metric. A working case is shown in Fig. 2(a), where this pixel-level metric successfully distinguishes a correct pair (left) from a wrong pair (right). Although this pixel-level metric can evaluate the stroke continuity in certain sense, this metric is very local, and could not differentiate correctly stitched strokes from the incorrect ones, and this metric may perform badly if the cut is near the boundary characters. A failure example is given in Fig. 2(b). The right pair in Fig. 2(b) is incorrect, but with relatively natural stroke transition, the right pair has a lower dissimilarity score. The left pair, unfortunately has two characters, "a" and "i" near the cutting boundary, the intensity change near the character's contour indicates big pixel-level transition discontinuity. Hence, its pixel-level dissimilarity score is higher than the right (incorrect) pair.

To increase our evaluator's robustness in document reassembly, we propose to use the higher-level text semantics to better estimate the local matching compatibility.

*2) Chararacter-Level Metric:* A natural text semantic feature for documents is the detected characters. If characters can be formed when two stripes are put together, then this stitching is more likely to be correct. Such a character-level feature suggests a higher-level feature than the aforementioned pixel-level evaluation. We build a character recognizer and used this feature to extract characters along the boundary of two stitched stripes.

Given a pair of stripes, $s_i$ and $s_j$, we consider a stitched fragment $(s_i, s_j)$ and extract characters near their shared boundary. We build our character detector using Google Tesseract [38] which can search a given image and report detected characters and their locations.

Specifically, we stitch two stripes together and perform character detection on it. For each detected character $u$, we use $f_c(u)$ to denote the recognition confidence (range $[0, 1]$). $U$ is a set containing detected characters, whose recognition confidences are larger than 0.7, on the stitching boundary. We use $d_c(s_i, s_j)$ to sum up the recognition confidences of all the detected characters,

$$d_c(s_i, s_j) = \sum_{u \in U} f_c(u). \tag{3}$$

We compute character-level compatibility score $C_c$ between a stripe pair $s_i$ and $s_j$ by

$$C_c(s_i, s_j) = 1 - \exp\left(-\frac{d_c(s_i, s_j)}{\alpha_2}\right), \tag{4}$$

where $\alpha_2$ is a scale factor. We set $\alpha_2 = 2$ in our experiments.

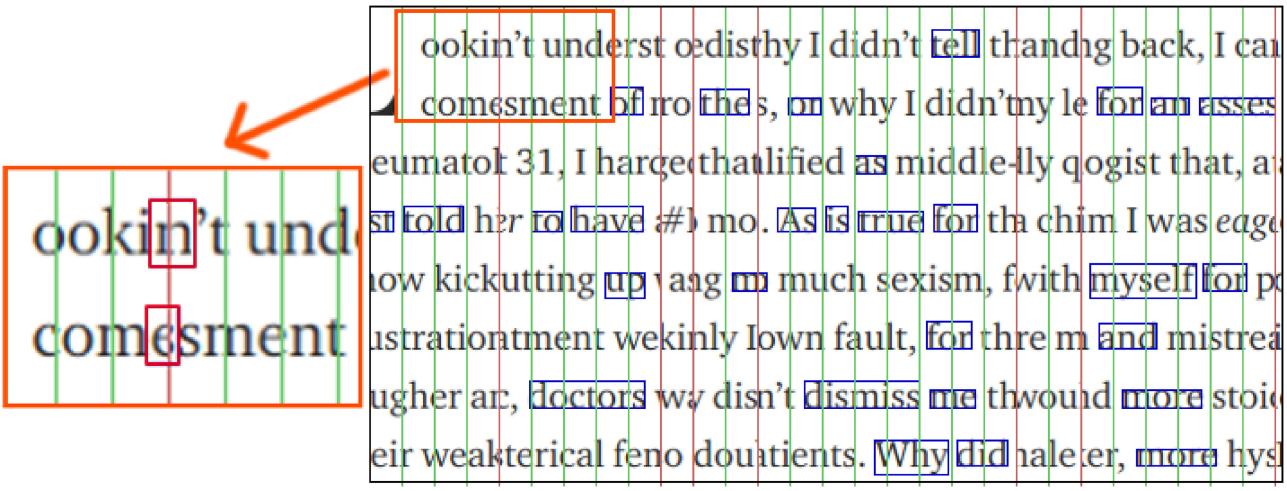


Fig. 3. A failure case of character-level metric and an illustration of word-level metric. The red rectangles in the zoomed-in region (on the left) indicate detected characters. The blue rectangles indicate the detected words. The red lines indicate the wrong composition between two stripes. The green lines indicate the correct composition between two stripes.

While this character metric does provide higher-level semantic information than the pixel-level metric, the character-level metric has two limitations. (1) First, character detection produces large amount of false positives. Many visually awkward compositions could be recognized as acceptable characters. An example is shown in Fig. 3. In the zoomed-in region, the identified characters "n" and "6" are both false positives. (2) Second, the character guided stitching still has significant ambiguity. In other words, incorrect matching pairs may still compose quite a few characters. This could lead to undesirable local minima in the subsequent global composition. Based on our observation in the global searching (to be discussed in the next section), we prefer this local estimator to be stricter and more accurate, so that it generates stitching candidate sequences with small false positives and little ambiguity, even it rejects some correct alignments. Therefore, we also build a higher-level and more reliable semantic metric.

*3) Mutual Low-Level Metric:* We consider pixel-level metric and character-level metric as *low-level metric*. By combining these two low-level metrics, we can obtain more reliable suggestions on potential stripe pairs. The compatibility scores of low-level metric $C_l$ is a linear combination of the pixel-level score $C_p$ and the character-level score $C_c$,

$$C_l = \lambda_0 * C_c + (1 - \lambda_0) * C_p, \tag{5}$$

where $\lambda_0$ is the relative weight of character-level metric.

We further improve the low-level metric to be less sensitive to local minima. At each step, between attaching $s_j$ to the right of $s_i$, we not only check whether $s_j$ is the best right neighbor of $s_i$, but also check whether $s_i$ is the best left neighbor of $s_j$. Specifically, this above metric is revised to construct a *mutual low-level metric* $C_m(s_i, s_j)$. Let $M_1(s_i, s_j)$ denote the normalized compatibility score over all acceptable right neighbors of $s_i$:

$$\begin{aligned} \text{Min}_i &= \min\{C_l(s_i, s_j)|j = 1 \cdots N\}, \\ \text{Max}_i &= \max\{C_l(s_i, s_j)|j = 1 \cdots N\}, \\ M_1(s_i, s_j) &= \frac{C_l(s_i, s_j) - \text{Min}_i}{\text{Max}_i - \text{Min}_i}. \end{aligned} \tag{6}$$

Similarly, let $M_2(s_i, s_j)$ be the normalized compatibility score over all acceptable left neighbors of $s_j$:

$$\begin{aligned} \text{Min}_j &= \min\{C_l(s_i, s_j)|i = 1 \cdots N\}, \\ \text{Max}_j &= \max\{C_l(s_i, s_j)|i = 1 \cdots N\}, \\ M_2(s_i, s_j) &= \frac{C_l(s_i, s_j) - \text{Min}_j}{\text{Max}_j - \text{Min}_j}. \end{aligned} \tag{7}$$

Finally, the mutual low-level metric is defined as

$$C_m(s_i, s_j) = M_1(s_i, s_j) + M_2(s_i, s_j), \tag{8}$$

and used as the low-level compatibility scores.

### B. High-Level Metric

*1) Word-Level Metric:* The word-level estimation is based on the detection of meaningful words from the text contents. Occasionally, for big stripes with small characters and short words, two stripes can compose valid words. But in most cases, composing meaningful words need multiple stripes. If following a specific order, a set of stripes composes multiple meaningful words across their stitching boundaries, we say that they form a valuable path of a certain compatibility score. A high-scored path suggests a more-likely correct permutation order of these stripes than a low-scored path.

Again, we build a word detector using the open-sourced Google Tesseract [38]. The word semantic detection is performed in a way similar to the character detection discussed in the previous section. We design a stochastic greedy algorithm

**Algorithm 1:** Generating Candidate Sequences

**Input:** Mutual low-level dissimilarity scores $M_l$.
**Output:** $M$ stripe sequences with length $L$.

```
Compute C_m, A, and B.
for k=1 to M do
    Randomly choose a stripe as the seed of sequence seq.
    for i = 1 to L do
        for stripe s in B[seq[i]] do
            if s was not selected and a random unit
              ε < A(seq[i], s) then
                Append stripe s to the end of seq.
                Break.
            end if
        end for
    end for
    Add seq to candidate sequences seq_set.
end for
return seq_set.
```

in generating $M$ random sequences of stripes, upon which the word detection is performed. Each sequence is stitched from $L$ stripes. This algorithm is elaborated in Section III-B2.

Figure 3 illustrates an example that such word features offer valuable information in evaluating local arrangements of stripes. In Fig. 3(b), the detected words are highlighted in blue boxes. These words suggest potential local composition paths among corresponding stripes. Note that the current OCR algorithm may not detect all the words, but with a large portion of words detected, this evaluator is still effective to suggest correct local alignments.

*2) Generating Candidate Sequences for Word Detection:* Word detection should be applied on a sequence of stitched stripes. We develop a generator that suggests different sequences that arrange stripes. The generator takes in two parameters $(L, M)$, then generates $M$ sequences of length $L$. In our experiments, we empirically set $L = N/4$ where $N$ is the total number of stripes in the puzzle.

At first, we randomly choose a stripe as the seed to start this sequence. Then we iteratively attach a suitable stripe to the right. Here we use the mutual low-level estimator $C_m$ (Eqn. 8) to help select the suitable right neighbors in each step. For example, starting from a randomly selected seed stripe $s_0$, we can select a stripe $s_1$ that has the highest compatibility score $C_m(s_0, s_1)$ as its right neighbor. Then, we continue to select $s_1$'s best right neighbor, until the sequence reaches the length of $L$. We can perform such a greedy sequence growing $M$ times to produce candidate sequences.

**Stochastic Greedy Selection:** We further improve this generation algorithm by adopting certain randomness into the greedy selection. Rather than always picking the best subsequent neighbor, we perturb our selection a little bit, in order to produce more potential candidate sequences. This is beneficial when the compatibility evaluation can not be very reliable, and we would like to give those not-the-top candidate neighbors some chances. To achieve this, we introduce an *acceptance rate* $A$ for each stripe pair $(s_i, s_j)$, which means that we have a probability of $A$ to accept this pair and a probability of $(1 - A)$ to reject it. $A$ is positively correlated to the relative mutual low-level metric scores $C_m$. Specifically, from $s_i$, if a candidate stripe $s_j$ can be attached to its right, and $C_m(s_i, s_j)$ is significantly higher than other stripes, then its acceptance rate is much higher too.

After the computation of mutual low-level metric, for each stripe $s_i$, we compute the acceptance rate of putting stripe $s_j$ to its right. We sort the compatibility scores in a descending order and discard the last $[f_0 * N]$ stripe pairs, where $N$ is the number of stripes and $f_0$ is a filtering rate parameter. When $f_0 = 0$, we keep all the stripe pairs. When $f_0 = 0.5$ we discard a half of stripe pairs. Let $B_i$ be the set of preserved stripe pairs and $Z_i$ be the score of the last element in $B_i$. Namely, $Z_i$ is the lowest accepted mutual low-level compatibility score.

For each stripe $s_j$ in $B_i$, we use $\beta$ to represent how much better the current pair is than $Z_i$,

$$\beta = U_a \left( \frac{C_m(s_i, s_j)}{Z_i} - 1 \right), \tag{9}$$

where $U_a$ is a scaling factor. Note that $Z_i$ has the smallest score in $B_i$, so $\beta$ is always non-negative. Finally, the acceptance rate $A$ for a pair $(s_i, s_j)$ is defined by

$$A(s_i, s_j) = \frac{\exp(\beta) - 1}{\exp(\beta) + 1}. \tag{10}$$

With $\beta \geq 0$, the value range of $A$ is $[0, 1)$.

We summarize our stripe sequence generation algorithm as follows. (1) Randomly select a stripe $s_i$ as the seed of a new sequence. (2) Iteratively select a new stripe $s_j$ to attach it to the end of the current sequence $[\ldots, s_i]$'s last element $s_i$ according $B_i$. (3) $s_j$ has a probability $A$ to be accepted or rejected. If accepted, proceed to next stripe; if rejected, try the next stripe in $B_i$. This procedure is summarized in Algorithm 1.

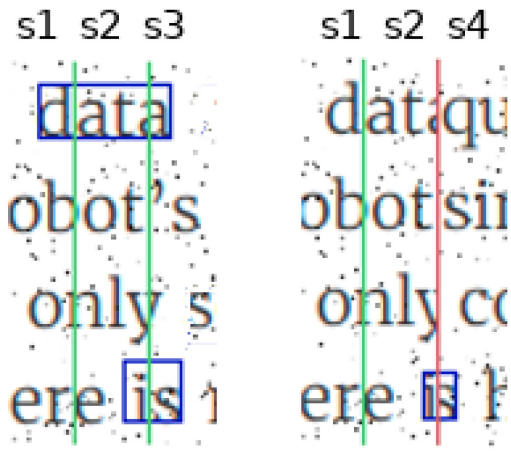
*3) Word-Path Compatibility Metric:* After We randomly generate $M$ candidate sequences, we perform an OCR word detection on each sequence. Then we use detected words to evaluate the compatibility of this sequence.

The *word-path* means that detected meaningful words could indicate a potential composition path among the stripes. Let $W = (w_1, w_2, \cdots, w_m)$ be the set of detected words. We use $len(w)$ to denote the number of stripes a word $w$ crosses, and $f_c(w)$ to denote the recognition confidence of the detected word $w$. We discard the detected words whose recognition confidences are less than 0.7 or the lengths are less than 3 in order to suppress false positives. For a candidate sequence $seq$, we define its **word-path compatibility score** $S_w(seq)$ as

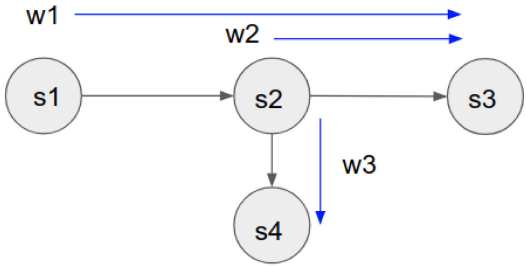
$$S_w(seq) = \sum_{w_k \in W(seq)} len(w_k) \times f_c(w_k). \tag{11}$$

Then for each stripe pair $(s_i, s_j)$, we can define their **pairwise word compatibility score** as

$$C_w(s_i, s_j) = \sum_{(s_i, s_j) \in seq} S_w(seq), \tag{12}$$




(a) Word detection on candidate sequences




(b) Corresponding word compatibility computation

Fig. 4. An example of word-path compatibility metric. (a) is a practical example. $w_1$, $w_2$, and $w_3$ are three detected words "data,""is," and "is" on sequences $(s_1, s_2, s_3)$, $(s_2, s_3)$, and $(s_2, s_4)$, respectively. Following Eqn. (12), $e(s_1, s_2) = f_c(w_1) * len(w_1) = 2.52$, $e(s_2, s_3) = f_c(w_1) * len(w_1) + f_c(w_2) * len(w_2) = 4.24$, and $e(s_2, s_4) = f_c(w_3) * len(w_3) = 1.52$. Thus, the correct alignment $e(s_2, s_3)$ has a higher compatibility score than the wrong alignment $e(s_2, s_4)$.

where $(s_i, s_j) \in seq$ means that the ordered pair $(s_i, s_j)$ appears in the sequence $seq$. We use $e(s_i, s_j)$ to estimate the likelihood of putting $s_i$ at the left of $s_j$. If $(s_i, s_j)$ appears in a high-confidence word path or in multiple word paths, then it is more likely to be correct.

With this we can build a *reassembly graph* $G = (S, E)$. The vertex set $S$ corresponds to all the stripes; the edge set $E$ contains directed edges $\{e(s_i, s_j)\}$ whose weights follows $C_w(s_i, s_j)$ in Eqn. (12). Fig. 4 gives an example to illustrate our design of the word-path compatibility metric.

### C. Compatibility Metric Summary

We propose a hierarchical metric to estimate the compatibility scores among stripes. The first step is to use pixel and character consistency to compute low-level compatibility scores $C_l$. We also make this low-level compatibility score less sensitive to local minima by introducing mutual low-level metric $C_m$. $C_m$ are used to generate candidate sequences for word detection. Using detected words and the corresponding stripe sequences, we build a reassembly graph $G$, whose vertex set is the stripes and the edge weights are pairwise word-path compatibility scores $C_w$.

## IV. Global Composition

After the computation of local alignment sequences, we obtain a reassembly graph $G = (S, E)$, where $S$ is the set of vertices that each corresponds to a stripe and $E$ is the set of edges that indicate the likelihood to an alignment between two stripes. Note that $(s_i, s_j) \neq (s_j, s_i)$ and $G$ is a directed graph. From $G$, we want to find an optimal Hamiltonian path that visits all, or as many as possible, vertices (stripes) exactly once, and maximizes the sum of weights of edges on the path. This reduces to a traveling salesman problem. As discussed in Section II, most existing solvers are greedy-based and sensitive to local minima.

We propose a novel integrated greedy composition and optimal matching (GCOM) algorithm, which consists of two steps: (1) Greedily compose stripes by word-path compatibility scores computed in Section III-B3. Locally optimal path is referred to as a *fragment*. Each stripe belongs to at most one fragment, and these fragments are separated. (2) With the locally composed fragments, search for an optimal path to splice these fragments together using the Kuhn-Munkres algorithm [14] and get the final reassembly. This algorithm is illustrated in Fig. 5.

### A. Greedy Composition

Using the word-path detector we designed, we extract a set of acceptable subsequences which compose meaningful words and compute the word-path compatibility scores as the weights' of edges among stripes. We greedily select highest weighed edge from them to compose fragments iteratively. Each stripe could belong to at most one fragment. This greedy composition will result in multiple disjoint fragments. These fragments will be spliced together by optimal matching in the next subsection.

### B. Optimal Matching

After the greedy merging discussed in Section IV-A, we obtained a set of fragments merged from stripes locally. Each stripe belongs to exactly one fragment, and each fragment implicitly encodes a directed path connecting its associated stripes. We can consider this as a new simplified reassembly graph, where each node corresponds to a group of stripes locally merged through a directed path. We want to connect these sub-paths together to form a Hamiltonian path. The number of fragments $N_f$ in this new graph is much smaller than the number of stripes $N$ in the original graph. Nevertheless, directly searching for the optimal Hamiltonian path in this new simplified graph is still NP-complete and prohibitive. However, after more carefully analyzing the structure of this fragment composition problem, we can see that this optimal sub-path connecting problem can actually be formulated into a simpler problem, which is solvable in polynomial time. This formulation can is described as follows.

The path in each fragment has two ends, a left end and a right end. To connect all these path to form a global reassembly, we shall always match a right end from one fragment with a left end from another fragment. Thus, this problem now reduces to an optimal assignment problem of a bipartite graph. A bipartite graph is a graph $G_{bi} = \{(U, V), E\}$ whose vertices can be divided into two disjoint and independent sets $U$ and $V$ such that

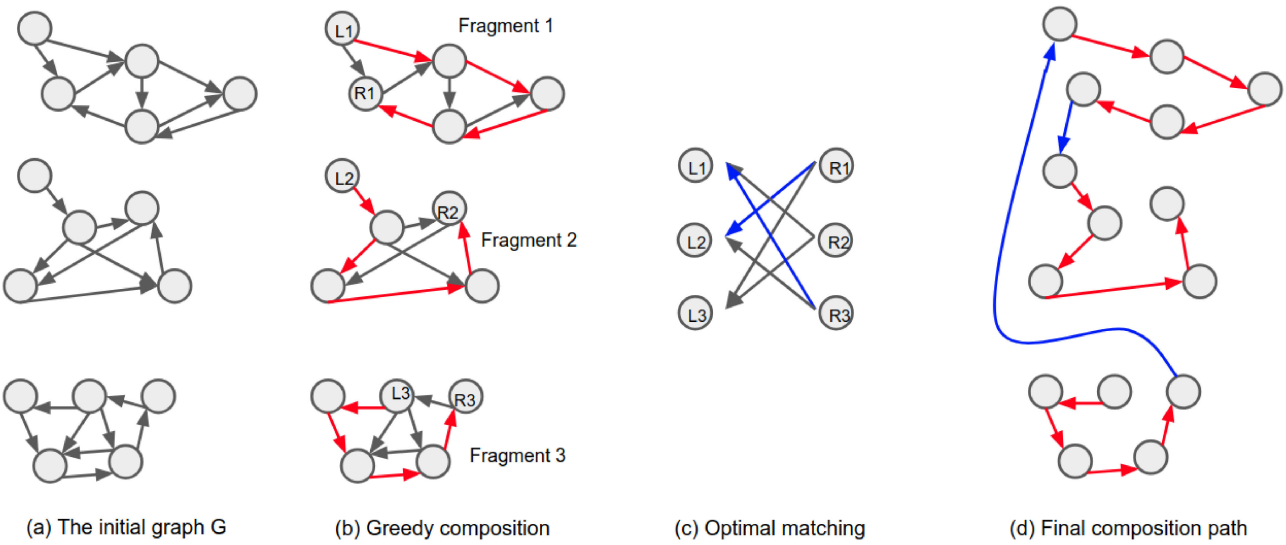



Fig. 5. An illustration of the GCOM algorithm. (a) is the initial graph produced by local alignments, pruned by word-path compatibility estimation. Each vertex corresponds to a stripe, and each edge $(s_i, s_j)$ represent a potential alignment that stitches stripe $s_j$ to the right of $s_i$. The weights of edges are computed by word-path metric. Note that we only render some edges, and omit many others. Based on word-path estimation, a greedy composition is performed and the result is given in (b). Red edges are the selected edges. Each fragment $F_i$ can be represented using its left most stripe $L_i$ and the right most stripe $R_i$ and $F_i$ implicitly encodes a local connecting path from $L_i$ to $R_i$. (c) Connecting all these local paths together reduces to finding an optimal matching on a bipartite graph $G_{bi}$, where each end can be selected at most once. The optimal matching suggested by Kuhn-Munkres algorithm are colored in blue. Combining all these paths together we obtain the final reassembly (d).

every edge connects a vertex in $U$ to one in $V$. We assign the left end stripes to $U$ and the right ends to $V$; then the edges $E$ between $U$ and $V$ are oriented from $V$ to $U$. If two ends are from the same fragment, there is no edge between them.

Because the number of the fragments $N_f$ is much smaller than the total number of stripes $N$. We enumerate the fragment pairs by stitching two fragments and reuse the word-path compatibility metric in Section III-B3. The weights $E$ in $G_{bi}$ is defined as the compatibility score between two fragments $f_1 \in V$ and $f_2 \in U$:

$$C_f(f_1, f_2) = \lambda_1 * C_w(f_1, f_2) + (1 - \lambda_1) * C_l(f_1, f_2), \quad (13)$$

where $\lambda_1$ is the relative weight of word-path metric.

We want to find an optimal assignment for this bipartite graph, namely, to select edges such that the sum of the edge weights is maximal and every node from $U$ connects to at most one node from $V$. The Kuhn-Munkres algorithm [14] is a combinatorial optimization algorithm that solves this problem in polynomial time $O(N_f^3)$. When we get the matchings of ends, we can compose these fragments into the final composition. The optimal matching algorithm is given in Algorithm 2

**Algorithm 2:** Optimal Matching

**Input:** A list of fragments $frags$.
**Output:** A global optimal composition $sol$.

```
Initialize: Empty arrays of U, V.
for each fragment in frags do
    Push the left end stripe into U
    Push the right end stripe into V.
end for
Build edges E from V to U.
res = KUHN-MUNKRES(U, V, E)
sol = SPLICE(frags, res)
return sol.
```

### C. Global Composition Summary

We build two graphs for document reassembly. The first graph $G$ is built in the previous section (see Section III), where the vertices are the stripes and the weights of the edges are the word-path compatibility scores. Then we apply a greedy composition on this graph to generate multiple fragments, each fragment encodes a directed path connecting its associated stripes. The second graph $G_{bi}$ is built for optimal matching. The vertices are the fragments and the weights of edges are computed by word-path compatibility scores by stitching two fragments together. The KM algorithm is applied on the bipartite graph $G_{bi}$ to obtain the final composition.

## V. Experiment Results

### A. Time Complexity Analysis

Suppose we have $N$ stripes, the time complexity of the pixel-level metric is $O(N^2 \times T_p)$, of the character-level metric is $O(N^2 \times T_c)$, and of the mutual low-level metric is $O(N^2 \times T_m)$. $T_p$, $T_c$ and $T_m$ are three constants related to different pairwise procedures. Let $T_l = T_p + T_c + T_m$, the time complexity of the low-level metric is $O(N^2 T_l)$.

For generating candidate sequences, we need to generate $M$ candidate sequences, whose lengths are $L$. As discussed in Section III-B2, a typical value for $L$ is $N/4$. The time complexity of generating candidate sequences is $O(\frac{1-f_0}{4} M N^2)$. For

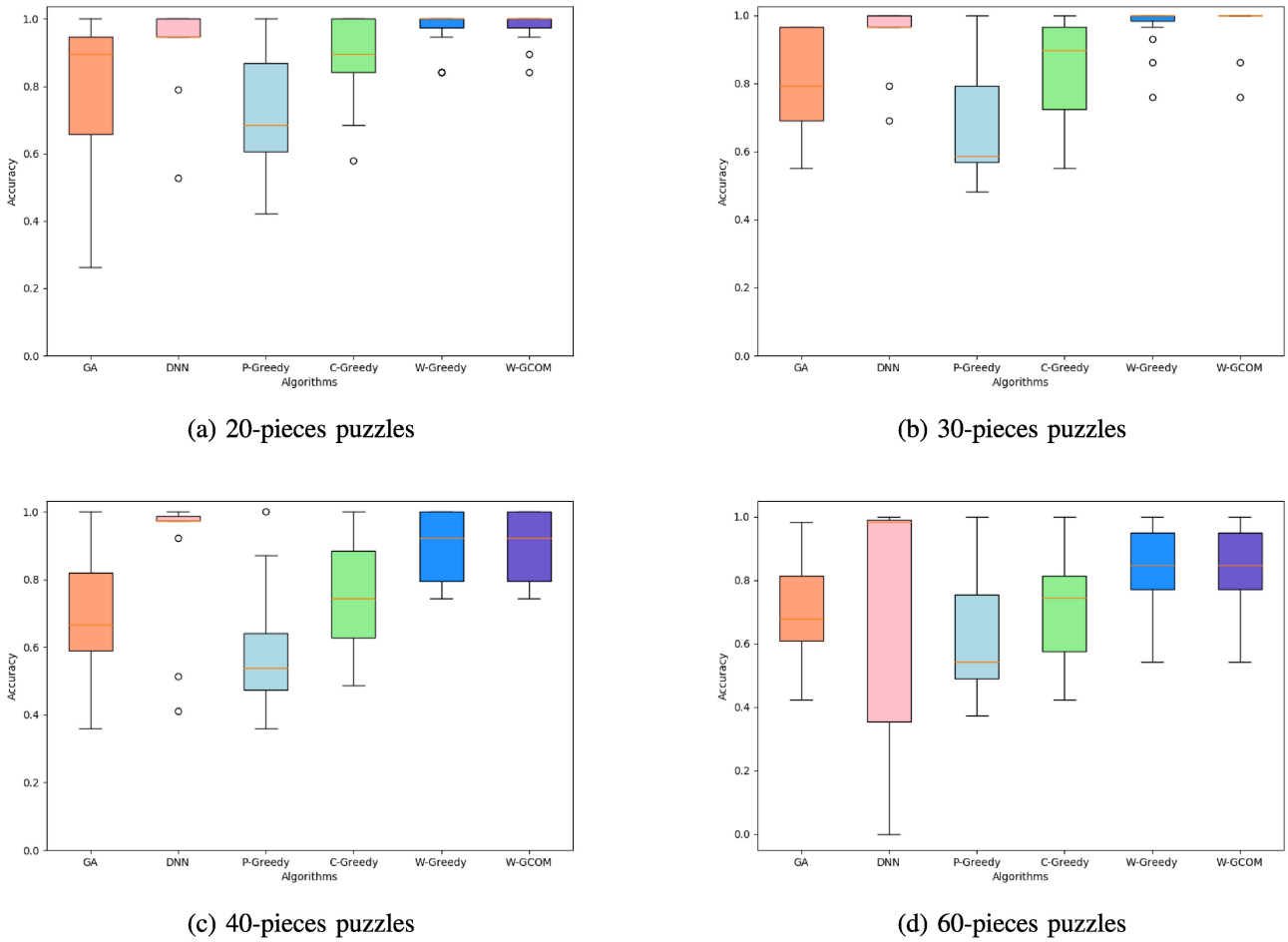



Fig. 6. Performance comparison for different algorithms running on synthetic stripe-cut documents from DocDataset. The four subplots show four stripe size ranges (difficult levels) of the tested shredded documents: 20, 30, 40, and 60 pieces. Each difficulty level contains 15 stripe puzzles. Documents are randomly downloaded from Internet and have English text printed in various font sizes and families. The $y$-axis shows the correct alignment ratio $NC$ (Eqn. 14). The $x$-axis lists different comparing methods: P-Greedy [37], GA [5], DNN [42], C-Greedy (ours), W-Greedy (ours), and W-GCOM (ours). The horizontal line in the box represents the median value. In (d) 60-pieces puzzles, the DNN method achieves 98.3% median accuracy while our W-GCOM has 79.7%. However, our W-GCOM is more robust than the DNN method that the average accuracy of the DNN is 65.5% while the average accuracy of our W-GCOM is 80.0%.

TABLE I
AVERAGE NC ACCURACY COMPARISON FOR DIFFERENT ALGORITHMS ON SYNTHETIC STRIPE-CUT DOCUMENTS FROM DOCDATASET

| #Stripes | GA [5] | DNN [42] | P-Greedy [37] | C-Greedy | W-Greedy (ours) | W-GCOM (ours) |
|---|---|---|---|---|---|---|
| 20 | 0.789 | 0.933 | 0.719 | 0.863 | 0.933 | **0.944** |
| 30 | 0.805 | 0.947 | 0.713 | 0.830 | 0.956 | **0.970** |
| 40 | 0.697 | **0.909** | 0.607 | 0.776 | 0.879 | 0.882 |
| 60 | 0.698 | 0.655 | 0.606 | 0.715 | 0.798 | **0.800** |

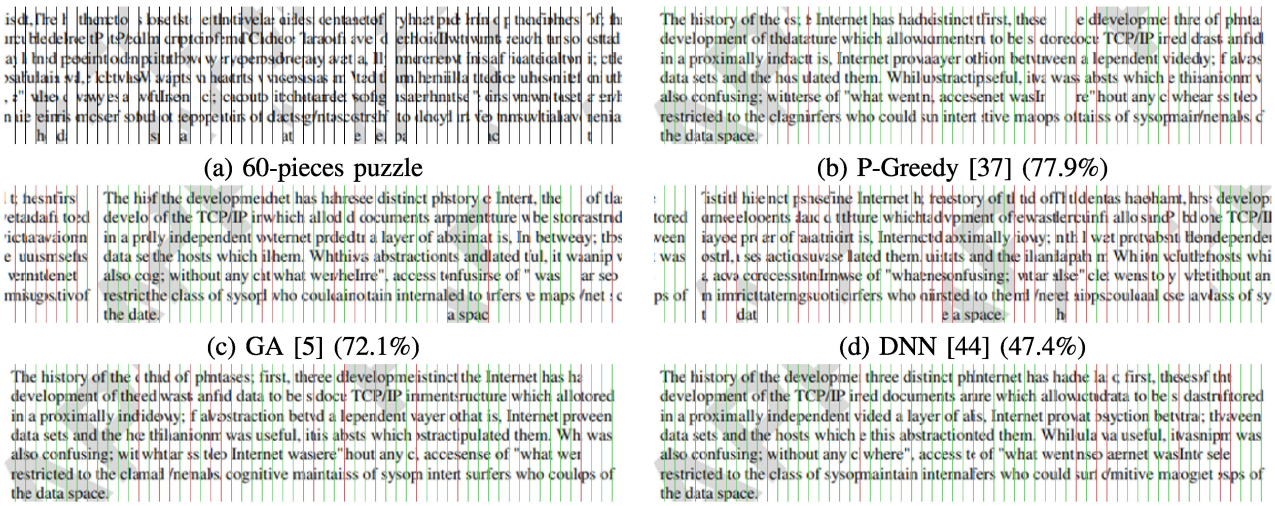
word-path metric, we apply OCR-word $M$ times on candidate sequences. We denote $O(T_w)$ to the computation complexity of the once OCR-word. Then the time complexity of the word-path metric is $O(M \times T_w)$. Therefore, the total time complexity of the proposed hierarchical compatibility metric could be simplified as $O(N^2 T_l + \frac{1-f_0}{4} MN^2 + MT_w)$.

The time complexity of the greedy composition is $O(NlgN)$. In our experiment, because the first graph $G$ is sparse, the actual computation time is much smaller than $O(NlgN)$. For the optimal matching part, the time complexity is $O(N_f^3)$, where $N_f$ is the number of merged fragments after greedy composition and $N_f \ll N$. So, the time complexity of the global composition part is $O(NlgN + N_f^3)$.

Overall, we solve the shredded document reassembly problem in polynomial time. The time complexity of the proposed pipeline is $O(N^2 T_l + \frac{1-f_0}{4} MN^2 + MT_w + NlgN + N_f^3)$.

### B. Dataset, Evaluation and Existing Strategies

We conduct thorough experiments on various stripe puzzles with different complexities to evaluate our algorithm and compare it with existing puzzle solving strategies. Since there is no public shredded document images dataset available, we create a new benchmark dataset and denote it as *DocDataset*. It contains 60 synthetic striped document puzzles and 3 physically striped document puzzles. 60 synthetic striped document puzzles are



(a) 60-pieces puzzle

(b) P-Greedy [37] (77.9%)

(c) GA [5] (72.1%)

(d) DNN [44] (47.4%)

(e) C-Greedy (77.9%)

(f) W-GCOM (ours, 84.7%)

Fig. 7. Reassembling a 60-piece puzzle of noisy digital document. The reassembly accuracy for each algorithm is indicated below each subfigure.

TABLE II
RUNTIME PERFORMANCE FOR DIFFERENT ALGORITHMS ON SYNTHETIC STRIPE-CUT DOCUMENTS FROM DOCDATASET

| #Stripes | GA [5] | DNN [42] | P-Greedy [37] | C-Greedy | W-Greedy (ours) | W-GCOM (ours) |
|---|---|---|---|---|---|---|
| 20 | 5s | 11s | 1s | 20s | 22s | 24s |
| 30 | 9s | 19s | 2s | 47 | 55s | 58s |
| 40 | 13s | 23s | 2s | 53 | 95s | 99s |
| 60 | 20s | 38s | 2s | 74s | 600s | 612s |

randomly generated, with four types of complexities of 20, 30, 40, and 60 stripes. Shredded documents contain randomly obtained English text contents, printed in various font sizes and font families. 3 physically striped document puzzles are generated by striped document shredders, then they are scanned and cropped into rectangle stripes manually.

To evaluate the reassembly accuracy, we utilize the neighbor comparison (NC) metric [2], which measures the ratio of correct alignments in the computed reassembly. Specifically, given a composition of $n$ stripes, this ratio is number of correct pairwise arrangement over $n-1$,

$$NC = \frac{\#\{\text{correct pairs}\}}{n-1}, \tag{14}$$

where $\#\{correct\ pairs\}$ denotes the number of correctly arranged pairs. In our figures, we use green seam lines to indicate correct alignments and red seam lines indicate wrong alignments. Intuitively, $NC$ is the ratio of green vertical seam lines over the total (green plus red) lines.

For comparative study, we classify state-of-the-art reassembly algorithms according to their compatibility metric and their composition strategies. On the compatibility metric, most existing algorithms build descriptors using image pixels [3], [5], [12], [26]–[37]. So we use the prefix *P* to denote this type of metric. In addition, we evaluate character-level metric for ablation study. We refer to the character-level metric as *C*. In contrast, our method uses a word-path metric, so we denote it using the prefix *W*. On the composition strategy, there are greedy growing and genetic based algorithms. The greedy growing methods, such as [4], [7], [9]–[13], [19], [23], [29], [32], [34], [37], are denoted by the suffix *Greedy*. The genetic algorithms, such as [5], [31], [35], are denoted by the suffix *GA*. Our Greedy Composition and Optimal Matching strategy is denoted by *GCOM*. Another recent state-of-the-art method [42] trains a deep neural network to measure the compatibility among stripes, we denote this method as *DNN*.

Hence, a comparison is performed on these different solving strategies: GA [5], P-Greedy [37], DNN [42], C-Greedy (for ablation study), W-Greedy (ours), and W-GCOM (ours). We implement our program in parallel on Intel Xeon(R) CPU E5-2630 v2(2.60 GHz $\times$ 24).

### C. Experiments on Synthetic Data

For synthetic data in DocDataset, we set $U_a = 2$, $\lambda_0 = 0.3$, $\lambda_1 = 0.5$, the discarding ratio $f_0 = 0.7$, and $M$ in Algorithm 1 to 150, 300, 1000, 8000 for 20-, 30-, 40-, 60-piece puzzles, respectively, to obtain sufficient sequences for word detection.

Figure 6 shows the performance comparison of these algorithms on puzzles of different complexities by box-plots. Box-plot is a method for graphically depicting groups of numerical data through their quartiles. The horizontal line in the box represents the median value. Lines extending vertically from the boxes indicate the upper and lower quartiles. Whiskers are plotted as circles indicating variability outside the upper and lower quartiles. Table I shows the average NC accuracy of these

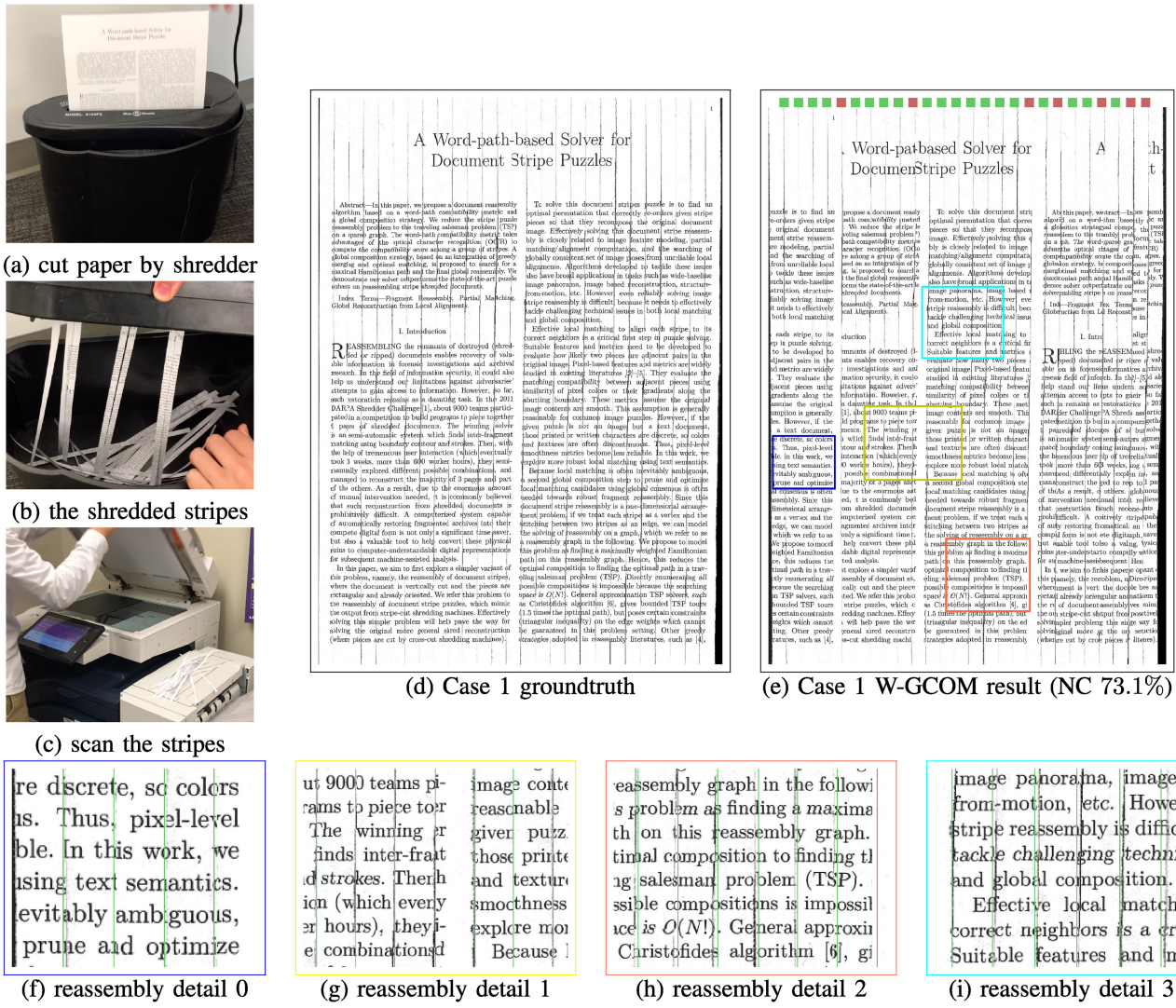


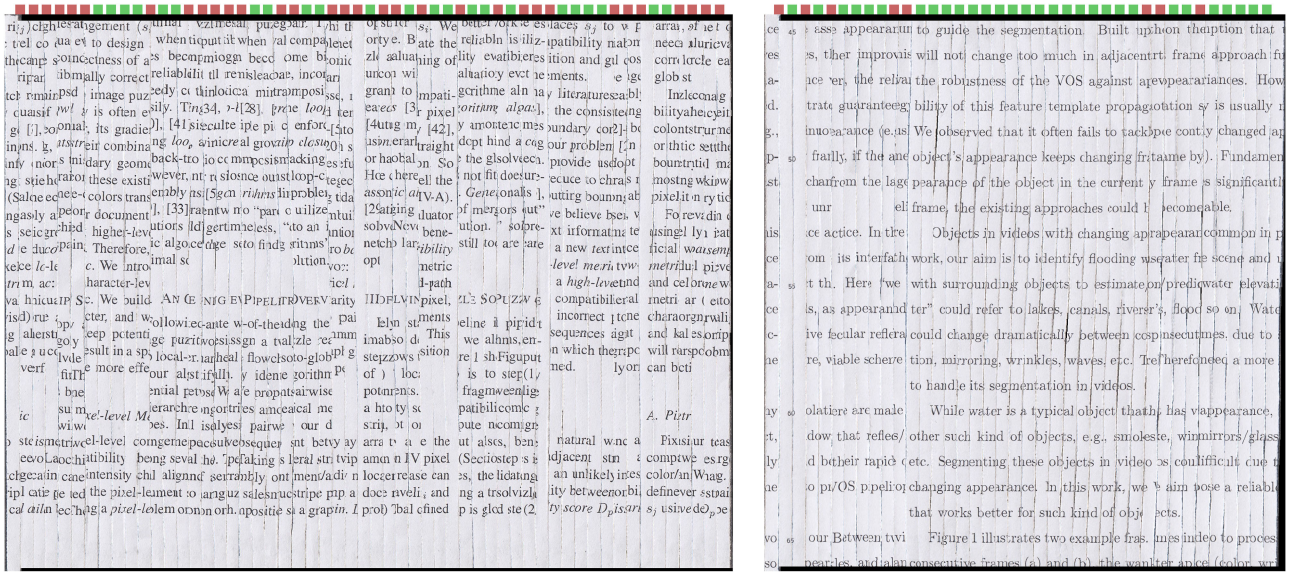

Fig. 8. Reassembling a physically shredded document. (a-c) Paper cutting and stripes scanning. (d) and (e) show the reassembly groundtruth and our reassembly result (W-GCOM), whose neighbor comparison accuracy is 73.1%. (f)–(i) illustrate some zoom-in details in (e).

TABLE III
COMPARISON WITH CONVENTIONAL METHODS ON THE PHYSICAL CASES FROM DOCDATASET

| Case | Stripes | DNN | P-Greedy | C-Greedy | W-Greedy | W-GCOM |
|---|---|---|---|---|---|---|
| 1 | 27 | 0.370 | 0.000 | 0.038 | 0.692 | **0.731** |
| 2 | 56 | **0.893** | 0.018 | 0.309 | 0.418 | 0.418 |
| 3 | 39 | 0.718 | 0.026 | 0.368 | 0.711 | **0.789** |

algorithms on puzzles of different complexities. GA and P-Greedy both use pixel-level metric to estimate the pairwise compatibility. We can see from experiments that because pixel-level metric is unreliable, the local minima found by GA and P-Greedy are often not very good. C-Greedy using character-level metric outperforms GA and P-Greedy using pixel-level metric because character-level metric provides more reliable guidance. However, when the number of stripes increase, character-level metric becomes ambiguous. False positives detected by the character-level metric hampers the performance of C-Greedy. DNN [42] utilizes deep neural networks to estimate pairwise compatibility. Although this method performs well when the number of stripes are small, this method obtains rather low accuracies in some cases (see circles in Fig. 6). The reason is that DNN is a pairwise metric which only considers the compatibility between two stripes. In contrast, our W-GCOM algorithm has stable performance because it takes sequences of stripes into account. Since the probability of sequence false positives is smaller than the pairwise false positives, when solving 60-stripe puzzles, the large number of stripes brings large number of pairwise false positives. Hence, the performance of the DNN method decays while the performance of W-GCOM is robust.

Specifically, on 60-stripe puzzles, the median value of the DNN [42] is 98.3%, while our W-GCOM is 79.7%. If the DNN



(a) Case 2 W-GCOM result (NC 41.8%) (b) Case 3 W-GCOM result (NC 78.9%)

Fig. 9. Reassembling the other two physically shredded document cases. (a) shows the W-GCOM result on the 56-stripe puzzle (Case 2). (b) shows the W-GCOM result on the 39-stripe puzzle (Case 3).

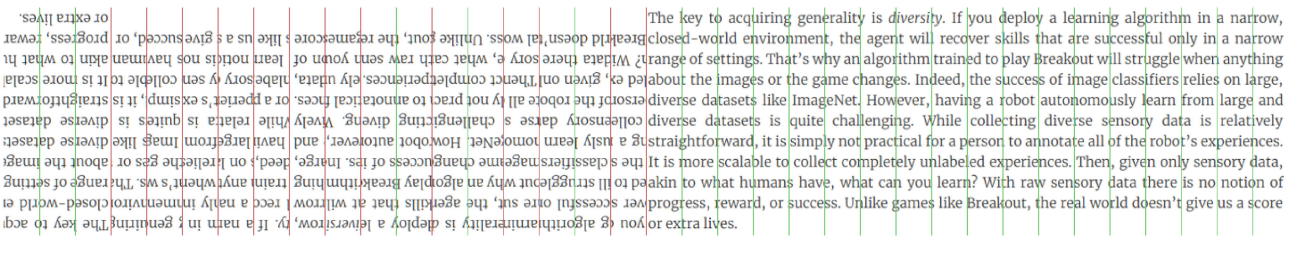


Fig. 10. Reassembling a randomly oriented stripe puzzle. The stripes are duplicated in the opposite directions. Our reassembly algorithm is directly performed on these combined stripes pool. Green line indicates correct matching and red line indicates wrong matching. The right part shows the fully correct reassembly result.

network has learned the stripe patterns, it can achieve better results than W-GCOM. However, in Table I, average accuracy for DNN is 65.5%, while the W-GCOM achieves 80.0%. There are much more severely failed results produced by the DNN, and the W-GCOM is more robust than the DNN. This is because the DNN method is built on a pairwise compatibility metric, while W-GCOM is built on the word-path metric, which is more robust to false positive stripe compositions because it takes sequences of stripes into consideration. Overall, the W-GCOM outperforms other state-of-the-art methods on the DocDataset.

Some qualitative comparisons of reassembly results from different algorithms are illustrated in Fig. 7. In our dataset, we incorporate some synthetic shredded documents with noise. Fig. 7 shows a 60-stripe puzzle of digital document with watermark. Our algorithm W-GCOM achieves the highest accuracy (79.7%) and the reassembly document is easy to read.

With the increasing puzzle complexity, the NC accuracy of pixel based algorithms gradually goes down. Because with more candidate stripes, there is a higher chance that pairs selected using pixel consistency are incorrect. For example, when the document is cut into 60 stripes, each stripe may only contain a small portion of letter, and the manual reassembly already becomes very difficult. When using the word-path metric, with the increased stripe number, we need to increase $L$ and $M$ in Algorithm 1 accordingly, so that sufficient amount and long enough stripe sequences can be generated for word detection. For example, when the stripes number is 60, we set $L = 15$ and $M = 8000$. Note that the number of all possible candidate stripe sequences in this case is $A_{60}^{15} \approx 10^{26}$, whose exhausting enumeration is prohibitive. As discussed in the above sections, we adopt the hierarchical filtering, to use the pixel-level and character-level metric to filter out impossible stripe pairs. Then we use stochastic greedy selection III-B2 to generate $M = 8000$ candidate sequences, which is much smaller than $10^26$ and will be applied in the OCR word detection to compute word-path compatibility.

Table II shows the runtime performance on the testing dataset. P-Greedy [37] runs fastest but has the worst performance. GA [5]

improves the results of P-Greedy but requires multithreading programming and fails on 60-stripe puzzles. Our method W-GCOM spends most time on OCR to detect meaningful characters and words from text images.

### D. Comparison on the Reassembly of Physical Shreds

Three real-shredding document reassembly cases are also tested in this work. We performed the experiments on two types of document shredders. One shredder takes the US letter paper, and cut it into 30 vertical stripes. The other one cuts an A4 paper into 60 stripes. After scanning these stripes and performing background removal, we obtain a set of rectangular digital stripes.

We compare P-Greedy [37], DNN [42], C-Greedy, our W-Greedy, and our W-GCOM in these physical cases. We set $M = 10,000$ in Algorithm 1 to obtain sufficient sequences for word detection. In addition, we set $\lambda_0 = 0.5, \lambda_1 = 0.7$, the OCR acceptance of confidence as 0.6, the discarding ratio $f_0 = 0.7$, and the $U_a = 1$ in Eqn. (9) to enlarge the search space of candidate sequences since noisy boundaries around stripes making pixel-level metric unreliable. For Case 2, we set length of candidate sequences $L = N/3$.

Figure 8(a) shows the groundtruth of the physical shred Case 1. We can see that there are irregular gaps between stripes. These noisy boundaries mislead the pixel-level metric and the character-level metric, so their accuracies are much lower than the word-path metric. Table III shows the NC accuracy among different methods. Case 1 is shown in Fig. 8. Case 2 and Case 3 are shown in Fig. 9. In the Case 1, the P-Greedy [37] has accuracy 0.00%, DNN [42] has the accuracy 37.0%, and the C-Greedy has accuracy 3.85%. They are much lower than the accuracy of 69.2% from the W-Greedy algorithm, and 73.1% from the W-GCOM algorithm. Note that both DNN [42] and our models are trained offline and tested on a comprehensive dataset. For practical usage, both results can be further improved if they are fine-tuned on specific text patterns and image resolutions. Fig. 8(e) shows the reassembly result by W-GCOM, the red squares and the greed squares in the top indicate the correct stripe pair and wrong stripe pair, respectively. Fig. 8(f)-(i) are four zoom-in details of Fig. 8(e). Different color box indicate different zoom-in regions. Fig. 9 shows the results of the other two physically shredded documents.

### E. Reassembling Randomly Oriented Stripes

In the previous sections, we formulated the problem in a way that assuming all the stripes are correct oriented. If the stripes are randomly oriented and some are upside down, the proposed solvers can still be applied directly. One way is to consider $n$ extra binary variables, indicating whether a $180^\circ$ rotation should be applied on each stripe. Another simpler approach is to generate a rotated version of each stripe, and put them together with the original stripes in a bigger pool of $2n$ pieces. Then perform our reassembly algorithm. Fig. 10 illustrates such an experiment. Since the OCR is not trained to recognize upside-down characters, our algorithm results in a composition that part of the document stripe puzzle produces fully correct reassembly (on the right). The other (upside-down) part (on the left) contains quite a few errors.

## VI. Conclusions

We present an algorithm to reassemble stripe-cut documents. The problem is formulated as solving a vertically cut stripe puzzle, and reduces to finding an optimal permutation that arranges these stripes back to the original document. The two technical contributions that lead to this effective pipeline are (1) the introduction of the text-based semantic feature, specifically, the *word-path metric* to evaluate the local matching compatibility, and (2) a novel integrated *Greedy-Composition and Optimal Matching (GCOM)* searching algorithm. Extensive experiments demonstrate that our algorithm significantly outperforms existing reassembly strategies.

*Limitations:* Even with the global composition, the reassembly results are still affected by the pairwise compatibility evaluation. When the number of stripes becomes very big, the word-path metric (W-) reaches its performance bottleneck, due to two reasons. (1) First, the OCR detector we used, Tesseract, is not perfect. For big puzzles where stripes are very small, many false detections appear. Two adjacent stripes may not compose a complete character and the pairwise character-level metric fails. Consequently, low level information from the character metric can not provide useful information for pruning in word detection. Eventually, only about 60% of words on the abutting boundaries are detected. (2) Second, another main parameter determining the word detection effectiveness is the length of candidate stripe sequences. A too short stripe sequence cannot yield robust word detection, because shorter words are less reliable and longer words only appear on long enough stripe sequences. However, generating/enumerating long stripe sequences is computationally expensive and poses a big search space. In addition, we observed that each running of Tesseract OCR detection involves significant initialization overhead. And this makes frequent calling of OCR detection quite inefficient. Currently, our sequence length is pre-determined, but ideally, this parameter needs to be adjusted according to the puzzle complexity and font sizes. To better tackle this problem, in the future we will explore more flexible or adaptive scheme for word detection. We will also train a new OCR detector based on not only words, but also root words and affixes, to further reduce the search space and build a more effective hierarchical detection scheme.

## Acknowledgment

The authors would like to express our deepest appreciation to anonymous reviewers whose comments helped improve and clarify this manuscript. Many thanks to Thiago Paixão for providing their experiment results. The authors gratefully acknowledge the effort of Xiaojin Huang for preparing physical shredded document dataset.

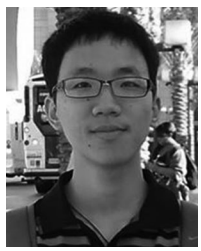
## References

[1] DARPA, "Darpa shredder challenge," 2011. [Online]. Available: http://archive.darpa.mil/shredderchallenge/.

[2] T. S. Cho, S. Avidan, and W. T. Freeman, "A probabilistic image jigsaw puzzle solver," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 183–190.

[3] D. Pomeranz, M. Shemesh, and O. Ben-Shahar, "A fully automated greedy square jigsaw puzzle solver," in *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2011, pp. 9–16.

[4] A. C. Gallagher, "Jigsaw puzzles with pieces of unknown orientation," in *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2012, pp. 382–389.
[5] D. Sholomon, O. David, and N. S. Netanyahu, "A genetic algorithm-based solver for very large jigsaw puzzles," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 1767–1774.
[6] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Manage. Sci. Res. Group, Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. 388, 1976.
[7] H. da Gama Leitao and J. Stolfi, "A multiscale method for the reassembly of two-dimensional fragmented objects," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 9, pp. 1239–1251, Sep. 2002.
[8] J. C. McBride and B. Kimia, "Archaeological fragment reconstruction using curve-matching," in *Proc. Comput. Vis. Pattern Recognit. Workshop*, 2003, vol. 1, Art. no. 3.
[9] E. Justino, L. S. Oliveira, and C. Freitas, "Reconstructing shredded documents through feature matching," *Forensic Sci. Int.*, vol. 160, no. 2, pp. 140–147, 2006.
[10] Q.-X. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann, "Reassembling fractured objects by geometric matching," in *Proc. ACM SIGGRAPH*, 2006, pp. 569–578.
[11] T. R. Nielsen, P. Drewsen, and K. Hansen, "Solving jigsaw puzzles using image features," *Pattern Recognit. Lett.*, vol. 29, no. 14, pp. 1924–1933, 2008.
[12] M. A. O. Marques and C. O. A. Freitas, "Reconstructing strip-shredded documents using color as feature matching," in *Proc. ACM Symp. Appl. Comput.*, 2009, pp. 893–894.
[13] S. Shang, H. T. Sencar, N. Memon, and X. Kong, "A semi-automatic deshredding method based on curve matching," in *Proc. IEEE Int. Conf. Image Process.*, 2014, pp. 5537–5541.
[14] J. Munkres, "Algorithms for the assignment and transportation problems," *J. Soc. Ind. Appl. Math.*, vol. 5, no. 1, pp. 32–38, 1957.
[15] H. Freeman and L. Garder, "Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 2, pp. 118–127, Apr. 1964.
[16] P. Butler, P. Chakraborty, and N. Ramakrishan, "The deshredder: A visual analytic approach to reconstructing shredded documents," in *Proc. IEEE Conf. Vis. Analytics Sci. Technol.*, 2012, pp. 113–122.
[17] A. Deever and A. Gallagher, "Semi-automatic assembly of real cross-cut shredded documents," in *Proc. 19th IEEE Int. Conf. Image Process. Conf.*, 2012, pp. 233–236.
[18] L. Zhu, Z. Zhou, and D. Hu, "Globally consistent reconstruction of ripped-up documents," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 1, pp. 1–13, Jan. 2008.
[19] F. Richter, C. X. Ries, N. Cebron, and R. Lienhart, "Learning to reassemble shredded documents," *IEEE Trans. Multimedia*, vol. 15, no. 3, pp. 582–593, Apr. 2013.
[20] K. Zhang and X. Li, "A graph-based optimization algorithm for fragmented image reassembly," *Graphical Models*, vol. 76, no. 5, pp. 484–495, 2014.
[21] H. Li, Y. Zheng, S. Zhang, and J. Cheng, "Solving a special type of jigsaw puzzles: Banknote reconstruction from a large number of fragments," *IEEE Trans. Multimedia*, vol. 16, no. 2, pp. 571–578, Feb. 2014.
[22] F. Richter, C. X. Ries, S. Romberg, and R. Lienhart, "Partial contour matching for document pieces with content-based prior," in *Proc. IEEE Int. Conf. Multimedia Expo.*, 2014, pp. 1–6.
[23] K. Zhang, W. Yu, M. Manhein, W. Waggenspack, and X. Li, "3-D fragment reassembly using integrated template guidance and fracture-region matching," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2138–2146.
[24] X. Li, K. Xie, W. Hong, and C. Liu, "Hierarchical fragmented image reassembly using a bundle-of-superpixel representation," *Comput.-Aided Geometric Des.*, vol. 71, pp. 220–230, 2019.
[25] C. Le and X. Li, "JigsawNet: Shredded image reassembly using convolutional neural network and loop-based composition," *IEEE Trans. Image Process.*, vol. 28, no. 8, pp. 4000–4015, Aug. 2019.
[26] E. Tsamoura and I. Pitas, "Automatic color based reassembly of fragmented images and paintings," *IEEE Trans. Image Process.*, vol. 19, no. 3, pp. 680–690, Mar. 2010.
[27] X. Yang, N. Adluru, and L. J. Latecki, "Particle filter with state permutations for solving image jigsaw puzzles," in *Proc. IEEE Comput. Vis. Pattern Recognit.*, 2011, pp. 2873–2880.
[28] F. A. Andalo, G. Taubin, and S. Goldenstein, "Solving image puzzles with a simple quadratic programming formulation," in *Proc. 25th SIBGRAPI Conf. Graph., Patterns Images*, 2012, pp. 63–70.
[29] B. Zhao, Y. Zhou, Z. Zhang, Y. Na, and T. Ma, "Information quantity based automatic reconstruction of shredded chinese documents," in *Proc. IEEE 26th Int. Conf. Tools Artif. Intell.*, 2014, pp. 1016–1020.
[30] K. Son, J. Hays, and D. B. Cooper, "Solving square jigsaw puzzles with loop constraints," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 32–46.
[31] Y.-F. Ge, Y.-J. Gong, W.-J. Yu, X.-M. Hu, and J. Zhang, "Reconstructing cross-cut shredded text documents: A genetic algorithm with splicing-driven reproduction," in *Proc. Annu. Conf. Genetic Evol. Comput.*, 2015, pp. 847–853.
[32] A. S. Atallah, E. Emary, and M. S. El-Mahallawy, "A step toward speeding up cross-cut shredded document reconstruction," in *Proc. 5th Int. Conf. Commun. Syst. Netw. Technol.*, 2015, pp. 345–349.
[33] D. Pöhler *et al.*, "Content representation and pairwise feature matching method for virtual reconstruction of shredded documents," in *Proc. IEEE 9th Int. Symp. Image Signal Process. Anal.*, 2015, pp. 143–148.
[34] T. Phienthrakul, T. Santitewagun, and N. Hnoohom, "A linear scoring algorithm for shredded paper reconstruction," in *Proc. IEEE 11th Int. Conf. Signal-Image Technol. Internet-Based Syst.*, 2015, pp. 623–627.
[35] Y.-J. Gong, Y.-F. Ge, J.-J. Li, J. Zhang, and W. Ip, "A splicing-driven memetic algorithm for reconstructing cross-cut shredded text documents," *Appl. Soft Comput.*, vol. 45, pp. 163–172, 2016.
[36] K. Son, J. Hays, and D. Cooper, "Solving small-piece jigsaw puzzles by growing consensus," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 1193–1201.
[37] J. Chen, D. Ke, Z. Wang, and Y. Liu, "A high splicing accuracy solution to reconstruction of cross-cut shredded text document problem," *Multimedia Tools Appl.*, vol. 77, no. 15, pp. 19 281–19 300, 2018.
[38] R. Smith, "An overview of the tesseract OCR engine," in *Proc. IEEE 9th Int. Conf. Document Anal. Recognit.*, 2007, vol. 2, pp. 629–633.
[39] Abbyy, Abbyy finereader, 2018. [Online]. Available: https://finereaderonline.com/en-us
[40] OnlineOCR, Online Ocr, 2018. [Online]. Available: https://www.onlineocr.net/
[41] N. Xing and J. Zhang, "Graphical-character-based shredded chinese document reconstruction," *Multimedia Tools Appl.*, vol. 76, no. 10, pp. 12 871–12 891, 2017.
[42] T. M. Paixao *et al.*, "A deep learning-based compatibility score for reconstruction of strip-shredded text documents," in *Proc. IEEE SIBGRAPI Conf. Graph., Patterns Images*, 2018, pp. 87–94.
[43] C. Le and X. Li, "Sparse3D: A new global model for matching sparse RGB-D dataset with small inter-frame overlap," *Comput. Aided Des.*, vol. 102, pp. 33–43, 2018.
[44] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5556–5565.



**Yongqing Liang** received the B.S. degree in computer science from Fudan University, Shanghai, China, in 2017. He is currently working toward the Ph.D. degree with the School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA, USA. His research interests include visual data understanding, computer vision, and computer graphics.

**Xin Li** (M'09–SM'15) received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2003, and the M.S. and Ph.D. degrees in computer science from Stony Brook University, Stony Brook, NY, USA, in 2008. He is currently an Associate Professor with the School of Electrical Engineering and Computer Science and the Center for Computation and Technology, Louisiana State University (LSU), Baton Rouge, LA, USA. He leads the Geometric and Visual Computing Laboratory with LSU. His research interests include geometric and visual data processing and analysis, computer graphics, and computer vision.