

CURSOR: Scalable Mixed-Order Hypergraph Matching with CUR Decomposition

Qixuan Zheng¹ Ming Zhang^{2*} Hong Yan¹

¹City University of Hong Kong

²Hong Kong Applied Science and Technology Research Institute (ASTRI)

{qixuzheng2-c, mzhang367-c}@my.cityu.edu.hk, h.yan@cityu.edu.hk

Abstract

To achieve greater accuracy, hypergraph matching algorithms require exponential increases in computational resources. Recent kd -tree-based approximate nearest neighbor (ANN) methods, despite the sparsity of their compatibility tensor, still require exhaustive calculations for large-scale graph matching. This work utilizes CUR tensor decomposition and introduces a novel cascaded second and third-order hypergraph matching framework (CURSOR) for efficient hypergraph matching. A CUR-based second-order graph matching algorithm is used to provide a rough match, and then the core of CURSOR, a fiber-CUR-based tensor generation method, directly calculates entries of the compatibility tensor by leveraging the initial second-order match result. This significantly decreases the time complexity and tensor density. A probability relaxation labeling (PRL)-based matching algorithm, specifically suitable for sparse tensors, is developed. Experiment results on large-scale synthetic datasets and widely-adopted benchmark sets demonstrate the superiority of CURSOR over existing methods. The tensor generation method in CURSOR can be integrated seamlessly into existing hypergraph matching methods to improve their performance and lower their computational costs.

1. Introduction

Finding the correspondences between a pair of feature sets by graph-matching has many applications in computer vision and pattern recognition tasks like feature tracking [11, 23, 39], image classification [36], object detection [24], and gene-drug association identification [5]. The second-order graph matching (pairwise matching) problem is a quadratic assignment problem (QAP), which is NP-hard [18]. Efforts to find soft-constraint approximate solutions

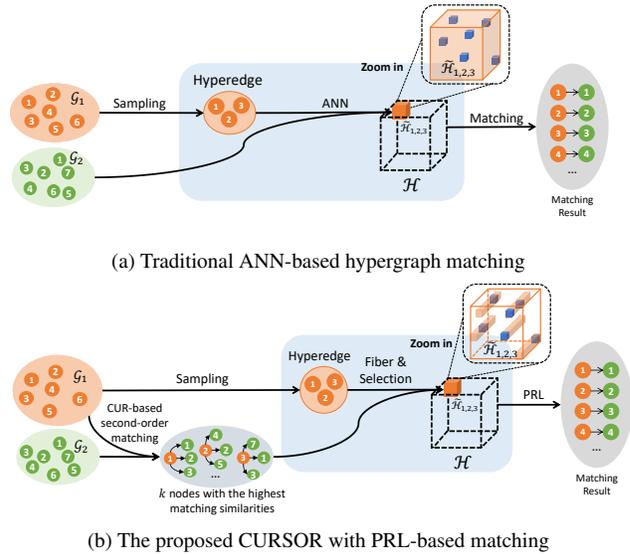


Figure 1. The comparison between the traditional ANN-based framework and CURSOR. Instead of calculating the whole tensor block (the light orange area in (a)) and extracting the highest compatibilities in each block (the blue cubes), CURSOR only calculates a small number of block fibers (the light orange area in (b)) and retains fewer elements in these fibers, effectively reducing computational costs for large-scale hypergraph matching. The method chooses the fibers based on the second-order matching result. CURSOR calculates fibers in all three tensor modes, and only one is shown in (b) for clarity.

[7, 8, 21, 32] have been limited to the representation of pairwise compatibilities.

Higher-order graphs, known as hypergraphs [9, 12, 19, 20, 30], integrate better geometric information and handle transformations like scaling and rotation better. The hypergraph matching problem considers the compatibility between two hypergraphs as a high-order tensor. The objective function aims to find the maximization over all permutations of the features. A kd -tree-based approximate near-

*Corresponding author.

est neighbor (ANN) method [9] to reduce the space and time complexity of the hypergraph matching algorithm has been used in many hypergraph matching algorithms (e.g. RRWHM [20], BCAGM [30] and ADGM [19]). ANN-based methods compute the sparse compatibility tensor by searching for nearest neighbors between randomly sampled hyperedges in the source graph and all the hyperedges in the target graph. Large-scale k^{th} -order hypergraph matching with n_2 points in the target graph has $O(n_2^k)$ time and space complexity for the compatibilities of each source graph hyperedge. Achieving a higher matching accuracy requires as many of the highest compatibilities as possible to increase the probability of finding the ground truth paired hyperedges, resulting in a denser compatibility tensor.

To address the above scalability issue, this work proposes a novel scalable hypergraph matching framework, CURSOR, based on cascaded mixed-order models with CUR decomposition. The comparison of CURSOR and traditional ANN-based methods is illustrated in Fig. 1. The traditional ANN-based methods construct the compatibility tensor directly with the nearest neighbors between the randomly sampled source hyperedges and all target ones. CURSOR first computes a roughly intermediate result with the proposed CUR-based second-order matching algorithm, drastically reducing the memory footprint of the compatibility matrices for large-scale graph matching. Subsequently, a small-scale target hyperedge subset, represented as third-order fiber tensors, can be generated utilizing the second-order results to calculate the compatibility tensor, substantially decreasing the computation complexity. The tensor generation method in CURSOR can integrate seamlessly into almost all existing state-of-the-art hypergraph matching algorithms [9, 12, 19, 20, 30] and significantly increase their matching performance at lower computational cost.

The contributions of this work are:

- We propose a novel cascaded second and third-order CUR-based hypergraph matching framework, CURSOR, to deal with large-scale problems. Under the same memory limitations, CURSOR can handle a more than ten times larger-scale hypergraph matching problem than current state-of-the-art algorithms.
- CURSOR contains a fiber-CUR-based compatibility tensor generation method using the rough matching result from the CUR-based second-order graph matching algorithm, which efficiently decreases the computational complexity and selects the proper sparse tensor entries.
- A PRL-based tensor matching algorithm is developed to significantly accelerate convergence during the matching process and increase the accuracy of matching.
- Experiment results show that CURSOR provides state-of-the-art matching accuracy by effectively finding the essential non-zero entries in the compatibility tensor.

2. Related Works

2.1. Hypergraph Matching

Second-order graph-matching algorithms [7, 8, 10, 21, 32] represented the geometric consistency between a pair of features as the edges of a graph to avoid ambiguities like repeated patterns and textures. These algorithms pursued an approximate solution as the problem is known to be NP-hard. Among them, Leordeanu and Herbert [21] and Cour *et al.* [8] estimated the rank-1 approximation of the compatibility matrix with power iteration as the flattening of the assignment matrix. Cho *et al.* [7] provided a novel random walk view for graph matching with a reweighting jump scheme and reduced the constraints in its iterative process. Recently, Wang *et al.* [32] proposed a functional representation for graph matching with the additional goal of avoiding the compatibility matrix construction. The performance of second-order methods was limited with the restriction to the normal graph embedding pairwise relationships.

In the past decade, to overcome the limitation of pairwise similarity, various hypergraph matching algorithms [9, 12, 19, 20] were developed based on the structural compatibilities between the higher-order hyperedges of two hypergraphs. In order to avoid the enormous computational cost of a full compatibility tensor, Duchenne *et al.* [9], extending the SM algorithm proposed by [21] to a higher order, constructed the sparse compatibility tensor with an ANN-based method, which is frequently used for generating compatibility tensors in later hypergraph matching works. Lee *et al.* [20] introduced the method of [7] to reweighted walk hypergraph matching problems, enforcing the matching constraint with a bi-stochastic normalization scheme. Lê-Huu and Paragios [19] decomposed the problem under different constraints and handled it with an alternating direction method of multipliers. Khan *et al.* [16] directly applied CUR decomposition to the full compatibility matrix and tensor at the cost of a higher space complexity than the ANN-based method. Although the ANN-based method can significantly decrease the time complexity during the matching process, it still has an enormous computational cost to calculate the compatibility tensor.

With the rise of deep learning, various learning-based graph-matching algorithms were proposed to learn the parameters as deep feature extraction hierarchies in a data-driven way [3, 4, 15, 26, 31, 34, 38]. Inspired by Wang *et al.* [33], Liao *et al.* [25] proposed the first unified hypergraph neural network, HNN-HM. Unlike the sparse compatibility tensor widely used in ANN-based algorithms, the dense deep feature matrices or tensors require more computation resources. Therefore, compared with ANN-based algorithms, the learning-based hypergraph matching algorithms can only handle much smaller-scale problems under the same memory constraint.

2.2. CUR Matrix and Tensor Decomposition

CUR decomposition [27] is used to compute the low-rank approximation of a matrix with the actual rows and columns. Assume a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. By selecting c columns and r rows from \mathbf{A} as $\mathbf{C} \in \mathbb{R}^{m \times c}$ and $\mathbf{R} \in \mathbb{R}^{r \times n}$, the low-rank approximation of \mathbf{A} can be formulated as $\mathbf{C}(\mathbf{C}^\dagger \mathbf{A} \mathbf{R}^\dagger) \mathbf{R}$, where \dagger is the pseudoinverse and $\mathbf{C}^\dagger \mathbf{A} \mathbf{R}^\dagger$ represents the matrix \mathbf{U} . Cai *et al.* [2] showed that if $\text{rank}(\mathbf{A}) < \min\{c, r\}$, \mathbf{U} can be directly represented by the intersection of \mathbf{C} and \mathbf{R} as its pseudoinverse. Xu *et al.* [37] proposed CUR+ to calculate the matrix \mathbf{U} with randomized matrix entries instead of the whole matrix when the matrix is not low rank. For tensor CUR decomposition, randomized fiber CUR decomposition can extend the CUR decomposition to the tensor [1]. The method first samples c_l fibers on mode- l and expands the fibers as matrix \mathbf{C}_l along mode- l . The intersection of all samples forms tensor \mathcal{R} and \mathbf{U}_l is the pseudoinverse of the mode- l expansion of \mathcal{R} . The fiber CUR decomposition of the tensor \mathcal{H} can be represented as:

$$\mathcal{H} \approx \mathcal{R} \times_{l=1}^k (\mathbf{C}_l \mathbf{U}_l) \quad (1)$$

where \times_l represents tensor times matrix along the l^{th} mode. The performance of the CUR decomposition is highly dependent on the selection of \mathbf{C} and \mathbf{R} (or \mathcal{R} in high order). Key samples can result in an approximation with high accuracy. Therefore, one of the most essential tasks of CUR decomposition is to find the required samples with less computation.

3. Method

3.1. Problem Setup

We follow the problem settings in [21]. Considering a pair of matched graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, the correspondences between \mathcal{G}_1 and \mathcal{G}_2 can be represented as a binary assignment matrix $\mathbf{X} \in \{0, 1\}^{n_1 \times n_2}$ where $n_1 = |\mathcal{V}_1|$ and $n_2 = |\mathcal{V}_2|$. To solve the NP-hard graph matching problem, we denote the elements of the assignment matrix with soft constraint as $X_{ij} \in [0, 1]$, where X_{ij} is the probability that the i^{th} node in \mathcal{V}_1 matches the j^{th} node in \mathcal{V}_2 . Following [9], we suppose every node in \mathcal{G}_1 matches exactly one node in \mathcal{G}_2 , and every node in \mathcal{G}_2 matches at most one node in \mathcal{G}_1 , i.e., $\forall i, \sum_j X_{ij} = 1$ and $\forall j, \sum_i X_{ij} \leq 1$. In the rest of the paper, we call \mathcal{G}_1 the source graph and \mathcal{G}_2 target graph.

The compatibility matrix of second-order graph matching, $\mathbf{H} \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, is the unfold of the fourth-order tensor $\hat{\mathcal{H}} \in \mathbb{R}^{n_1 \times n_2 \times n_1 \times n_2}$ where $\hat{H}_{i_1, j_1, i_2, j_2}$ represents the similarity between edges (i_1, i_2) and (j_1, j_2) . The soft-constraint assignment matrix is flattened as \mathbf{x} . The second-order graph-matching problem can be represented as the optimization of the function:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{x}^T \mathbf{H} \mathbf{x} \\ \text{s.t.} \quad & \sum_{j \in \text{ind}_i} x_j = 1, \forall i \end{aligned} \quad (2)$$

where $\text{ind}_i = \{(i-1)n_2 + 1, \dots, in_2\}$ represents the index set of the i^{th} row of \mathbf{X} . The compatibility matrix is symmetric, which means the compatibility between (i_1, i_2) and (j_1, j_2) is the same as that between (i_2, i_1) and (j_2, j_1) . Equation (2) can be cast into a classical Rayleigh quotient problem, and \mathbf{x} is proved to be associated with the main eigenvector of \mathbf{H} [9], which can be calculated with methods such as power iteration.

The k^{th} -order hypergraph-matching problem can be extended to:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathcal{H} \otimes_1 \mathbf{x} \otimes_2 \dots \otimes_k \mathbf{x} \\ \text{s.t.} \quad & \sum_{j \in \text{ind}_i} x_j = 1, \forall i \end{aligned} \quad (3)$$

where the k^{th} -order supersymmetric tensor \mathcal{H} represents the compatibility between the hyperedges in the two graphs [9]. \otimes_l is the mode- l product of the tensor and vector, which is calculated as:

$$(\mathcal{H} \otimes_l \mathbf{x})_{i_1, \dots, i_{l-1}, i_{l+1}, \dots, i_k} = \sum_{i_l} H_{i_1, \dots, i_l, \dots, i_k} x_{i_l} \quad (4)$$

3.2. CUR-based Second-Order Graph Matching

Dealing with large-scale graph matching problems, with thousands of paired nodes, is not feasible due to the terabytes of computer memory required for the compatibility matrix \mathbf{H} . CURSOR estimates the low-rank approximation of the compatibility matrix with CUR decomposition. Instead of directly generating the whole matrix with huge memory usage, it calculates a small number of rows and columns. Because of the symmetric property of the compatibility matrix, the column sampling is sufficient to decrease the computational complexity. By randomly selecting c columns from \mathbf{H} as \mathbf{C} , the second-order compatibility matrix can be decomposed into two smaller-sized matrices $\mathbf{C} \in \mathbb{R}^{n_1 n_2 \times c}$ and $\mathbf{U}_* \in \mathbb{R}^{c \times c}$. Following CUR+ [37], \mathbf{U}_* is calculated as:

$$\mathbf{U}_* = \arg \min_{\mathbf{U}} \|R_\Omega(\mathbf{H}) - R_\Omega(\mathbf{C} \mathbf{U} \mathbf{C}^T)\|_F \quad (5)$$

where $R_\Omega(\cdot)$ is the symbol used in the original work of CUR+ [37] to represent the matrix entries, including the randomly selected entries and all the intersection elements of \mathbf{C} and \mathbf{C}^T . $\|\cdot\|_F$ represents the Frobenius norm of the matrix. The multiplication of $\mathbf{H} \mathbf{x}$ to update \mathbf{x} in every iteration can be simplified as:

$$\mathbf{H} \mathbf{x} \approx \mathbf{C} \mathbf{U}_* (\mathbf{C}^T \mathbf{x}) \quad (6)$$

which reduces the time and space complexity in every iteration from $O(n_1^2 n_2^2)$ to $O(cn_1 n_2)$. For large-scale graph matching problems, $c \ll n_1 n_2$.

The assignment matrix, \mathbf{X} , is calculated with the CUR decomposition of \mathbf{H} by applying a soft-constraint second-order graph-matching algorithm, like SM [21] or RRWM [7]. For $i \in \{1, \dots, n_1\}$, the k entries with highest probabilities in $\mathbf{X}_{i,:}$ are found as the best k match set $\mathcal{P}_i^k = \{s_{ij}\}_{j=1}^k$, where s_{ij} represents the index of the j^{th} highest probability in $X_{i,:}$. The detailed algorithm is shown in Algorithm 1. The CUR-based method may lead to lower matching accuracy with fewer sampled rows and columns if only second-order matching is applied. However, it can provide a rough result to follow with higher-order graph matching, thereby avoiding the infeasible large-scale computations.

Algorithm 1 CUR-based second-order graph matching

Input Point sets P_1, P_2 with size n_1, n_2 , column indices I , entry indices $J \supseteq I, k$
Output Best k match set $\mathcal{P}^k = \text{set}(\mathcal{P}_1^k, \dots, \mathcal{P}_{n_1}^k)$.
 $\mathbf{C} \leftarrow \mathbf{H}(:, I)$ {calculate columns with P_1, P_2 }
 $\mathbf{U}_* \leftarrow \text{CUR}(\mathbf{C}, \mathbf{H}(J, J))$ {Based on Eq. (5).}
 $\mathbf{x} \leftarrow \text{Matching}(\mathbf{C}\mathbf{U}_*\mathbf{C}^T)$
 $\mathbf{X} \leftarrow \text{reshape}(\mathbf{x})$
for $i = 1, \dots, n_1$ **do**
 {Find k entries with highest probabilities for $X(i, :)$ }
 $\mathcal{P}_i^k \leftarrow \{s_{ij}\}_{j=1}^k$
end for

3.3. Fiber-CUR-based Tensor Generation

The ANN-based method searches for the highest compatibilities in all target hyperedges, which is time and space-consuming for large-scale hypergraph matching. Based on randomized fiber CUR decomposition, the third-order compatibility tensor can be generated with the pairwise graph-matching result to reduce the computational cost. The second-order matching result from Algorithm 1 is \mathcal{P}^k , where $k \ll n_2$. The method does not need to compare the hyperedge $(i_1, i_2, i_3) \in \mathcal{E}_1$ with all the hyperedges in the target hypergraph, but with the hyperedge set $\{(j_1, j_2, :), (j_1, :, j_3), (:, j_2, j_3)\} \in \mathcal{E}_2$ where $j_1 \in \mathcal{P}_{i_1}^k, j_2 \in \mathcal{P}_{i_2}^k$, and $j_3 \in \mathcal{P}_{i_3}^k$. The r highest entries from each hyperedge's calculated compatibilities in the source graph are selected. Since each hyperedge is compared with fewer target hyperedges, r is much smaller than the number of selected compatibilities r_1 in [9].

The light blue areas in Fig. 1 illustrate the comparison between the prior ANN-based tensor generation methods and the proposed fiber-CUR-based method in terms of the tensor structure. The compatibility between hyperedge (i_1, i_2, i_3) in the source graph and all the hyperedges

in the target graph can be represented as the tensor block $\mathcal{H}_{i_1, i_2, i_3} = H_{\text{ind}_{i_1}, \text{ind}_{i_2}, \text{ind}_{i_3}}$ (orange cubes in Fig. 1). Traditional ANN-based methods first randomly select t tensor blocks, then calculate each tensor block with n_3^3 time and space complexity and reserve the r_1 highest entries in each block. In CURSOR, with the second-order graph matching result \mathcal{P}^k , only the corresponding fibers of each tensor block, \mathcal{C} are estimated, and the r highest entries are selected. Consequently, the algorithm's complexity reduces from $O(tn_3^3 + tr_1)$ to $O(tk^2 n_2 + tr)$ where $r \ll r_1$. As only the highest compatibilities matter, there is no need to calculate the redundant \mathbf{U} and \mathcal{R} in Eq. (1). Compared with the original randomized fiber CUR decomposition, the compatibility tensor requires less computation. The fibers in \mathcal{C} are selected based on \mathcal{P}^k rather than by random sampling and are closer to the key samples. Therefore, the ground truth matching compatibility can be located with a higher probability, resulting in higher matching accuracy. The total algorithm is given as Algorithm 2.

Algorithm 2 Fiber-CUR-based tensor generation

Input Point sets P_1, P_2 with size n_1, n_2 , best k initial guess set $\mathcal{P}^k = \text{set}(\mathcal{P}_1^k, \dots, \mathcal{P}_{n_1}^k)$
Output Sparse tensor \mathcal{H}
 $\mathcal{H} \leftarrow$ empty tensor
 $\mathcal{I} \leftarrow$ hyperedges randomly sampled from P_1
for $i = (i_1, i_2, i_3) \in \mathcal{I}$ **do**
 $e \leftarrow \text{computeHyperedgeFeature}(i, P_1)$
 $\mathcal{F} \leftarrow$ empty feature set
 {Calculate corresponding fibers in all directions.}
 $\mathcal{J}_1 \leftarrow \{(1 : n_2, j_2, j_3)\}, j_2 \in \mathcal{P}_{i_2}^k, j_3 \in \mathcal{P}_{i_3}^k$
 $\mathcal{J}_2 \leftarrow \{(j_1, 1 : n_2, j_3)\}, j_1 \in \mathcal{P}_{i_1}^k, j_3 \in \mathcal{P}_{i_3}^k$
 $\mathcal{J}_3 \leftarrow \{(j_1, j_2, 1 : n_2)\}, j_1 \in \mathcal{P}_{i_1}^k, j_2 \in \mathcal{P}_{i_2}^k$
 $\mathcal{J} = \text{set}(\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3)$
 for $j \in \mathcal{J}$ **do**
 $f \leftarrow \text{computeHyperedgeFeature}(j, P_2)$
 $\mathcal{F} \leftarrow \mathcal{F} \cup f$
 end for
 $\mathcal{S} \leftarrow$ search for r highest similarities(\mathcal{F}, e)
 for $s \in \mathcal{S}$ **do**
 $\text{ind}(i, j_s) \leftarrow$ index of $\mathcal{H}(i, \text{index}(s))$
 $\mathcal{H}(\text{ind}(i, j_s)) \leftarrow$ compatibility(e, s)
 end for
end for

3.4. PRL-based Matching Algorithm

To accelerate the convergence of the matching process, a fast hypergraph matching algorithm was developed based on probabilistic relaxation labeling (PRL) that takes advantage of the high sparsity of the compatibility tensor. The original PRL algorithm [13] updates the probability that a label is assigned to an object with a set of specified com-

patibilities, and has been applied in several graph matching problems [6, 28, 35]. For each $i \in \mathcal{G}_1$ and $j \in \mathcal{G}_2$, $p_i(j)$ represents the probability that i is associated with j , which can be updated according to the following equation:

$$p_i^{(k+1)}(j) = \frac{p_i^{(k)}(j)[1 + q_i^{(k)}(j)]}{\sum_m p_i^{(k)}(m)[1 + q_i^{(k)}(m)]} \quad (7)$$

with

$$q_i^{(k)}(j) = \sum_{l=1}^{n_1} d_{il} \left[\sum_{m=1}^{n_2} r_{il}(j, m) p_l^{(k)}(m) \right] \quad (8)$$

where d_{il} is a weight factor representing the influence of l on i and $\sum_l d_{il} = 1$. The factor $r_{il}(j, m) \in [-1, 1]$ denotes the relationship between the pairwise feature of edge $(i, l) \in \mathcal{E}_1$ and $(j, m) \in \mathcal{E}_2$.

In our work, define $R_{il}(j, m) = 0.5(r_{il}(j, m) + 1) \in [0, 1]$ and set $d_{il} = n_1^{-1}$. By replacing the weighting factor $p_i^{(k)}(j)$ in Eq. (7) with updated probabilities, it gives

$$p_i^{(k+1)}(j) = \frac{[\sum_l \sum_m R_{il}(j, m) p_l^{(k)}(m)]^2}{\sum_i [\sum_l \sum_m R_{il}(j, m) p_l^{(k)}(m)]^2} \quad (9)$$

which contributes to a more consistent result and faster convergence. Define

$$R_{i_1 i_2}(j_1, j_2) = \begin{cases} M_{i_1, j_1} & i_1 = i_2 \text{ and } j_1 = j_2 \\ \hat{H}_{i_1, j_1, i_2, j_2} & \text{otherwise} \end{cases} \quad (10)$$

where $\hat{H}_{i_1, j_1, i_2, j_2}$ is the corresponding value in the compatibility matrix \mathbf{H} and M_{i_1, j_1} is related to the first-order compatibility between node $i_1 \in \mathcal{G}_1$ and $j_1 \in \mathcal{G}_2$. Since $\hat{H}_{i, j, i, j} = 0$ for all i, j , the numerator of Eq. (9) can be updated as:

$$\hat{p}_i^{(k+1)}(j) = (M_{i, j} p_i^{(k)}(j) + \sum_l \sum_m \hat{H}_{i, j, l, m} p_l^{(k)}(m))^2 \quad (11)$$

Denote δ as $\sum_l \sum_m \hat{H}_{i, j, l, m} p_l^{(k)}(m)$. To increase the corresponding true matching $p_{i_0}^{(k)}(j_0)$ consistently during every iteration, $\hat{p}_{i_0}^{(k+1)}(j_0)$ must satisfy $(M_{i_0, j_0} p_{i_0}^{(k)}(j_0) + \delta)^2 \geq p_{i_0}^{(k)}(j_0)$, which leads to

$$\delta \geq 0.25 M_{i_0, j_0}^{-1} \quad (12)$$

If Eq. (12) is guaranteed, the probabilities can converge fast. The detailed derivation of the PRL-based method can be found in the supplement.

To extend the algorithm to third-order hypergraph matching, replace the probability set $\mathbf{p} = \{p_i(j)\}$ with the vector \mathbf{x} , the column-wise flattening of soft-constraint assignment matrix \mathbf{X} . After normalization, rewrite Eq. (11) as

$$\hat{\mathbf{x}}^{(k+1)} = (\alpha \hat{\mathbf{m}} \odot \mathbf{x}^{(k)} + (1 - \alpha) \mathcal{H} \otimes_1 \mathbf{x}^{(k)} \otimes_2 \mathbf{x}^{(k)})^2 \quad (13)$$

where \odot is the element-wise multiplication and $\alpha \in [0, 1]$ is a balance weight between the first and third-order compatibilities. The square calculation in Eq. (13) is also element-wise. The first-order compatibility vector $\hat{\mathbf{m}}$ is obtained by column-wise flattening $\hat{\mathbf{M}}$. The steps are shown in Algorithm 3. For a tensor with high sparsity, the non-zero entries of $\mathcal{H} \otimes_1 \mathbf{x}^{(k)} \otimes_2 \mathbf{x}^{(k)}$ are concentrated. Therefore $\mathbf{x}^{(k)}$ has a fast convergence speed if the ground truth compatibility entries are selected in the sparse tensor. According to Eq. (12), a lower α can be set for the tensor with high sparsity to achieve faster convergence.

Algorithm 3 PRL-based hypergraph matching

Input Sparse compatibility tensor \mathcal{H} , initial assignment matrix \mathbf{X} , first-order compatibility vector $\hat{\mathbf{m}}$, α
Output Soft-constraint assignment matrix \mathbf{X}
repeat
 $\mathbf{x} \leftarrow \text{flatten}(\mathbf{X})$
 $\delta \leftarrow \mathcal{H} \otimes_1 \mathbf{x} \otimes_2 \mathbf{x}$
 $\mathbf{x} \leftarrow \alpha \hat{\mathbf{m}} \odot \mathbf{x} + (1 - \alpha) \delta$
 $\mathbf{x} \leftarrow \mathbf{x} \odot \mathbf{x}$
 $\mathbf{X} \leftarrow \text{reshape}(\mathbf{x})$
 $\mathbf{X} \leftarrow \text{norm}(\mathbf{X})$ {Normalize across columns}
until \mathbf{X} converges

4. Experiments

CURSOR was compared with four learning-free third-order ANN-based hypergraph matching algorithms: Tensor Matching (TM) [9], Hypergraph Matching via Reweighted Random Walks (RRWHM) [20], BCAGM in third-order (BCAGM3) [30], and Alternating Direction Graph Matching (ADGM)[19]. The experiments were conducted on the original implementations provided by their authors. The proposed tensor generation method was integrated into each of these algorithms, represented as CURSOR+TM/RRWHM/BCAGM3/ADGM in the experiments. The all-ones vector was set as the starting point, and the Hungarian algorithm [17] turned the output into a proper matching. CURSOR was also compared with the state-of-the-art deep-learning-based algorithm HNN-HM [25] on the House and Hotel dataset, which is relatively small-scale. Since HNN-HM failed for datasets with $n_1 > 40$ under the same memory constraint, it was not compared with CURSOR on other datasets. The hyperparameter α in Algorithm 3 was set to 0.2 during the experiments. The experiments were run on a computer with an Intel Core i7-9700 CPU @ 3.00 GHz and 16 GB of memory. All quantitative results were obtained by 50 trials. Due to space limitations, the ablation studies and parameter sensitivity analysis are given in the supplement.

The compatibility features for each order and the param-

				ADGM			CURSOR				
n_1	n_2	σ	t	r_1	Memory (\mathcal{H})	Accuracy	c	k	r	Memory ($\mathbf{H} + \mathcal{H}$)	Accuracy
30	30	0.02	900	900	89.35MB	1	15	5	5	0.62MB	1
30	50	0.02	1500	2500	413.88MB	1	15	5	5	0.82MB	1
50	50	0.02	2500	2500	701.80MB	1	20	7	7	2.37MB	1
50	100	0.02	5000	10000	5.45GB	0.974	100	10	10	9.62MB	0.982
100	100	0.02	10000	10000	11.28GB	1	100	15	20	30.65MB	1
300	300	0.01	30000	-*	-	-	200	20	20	215.92MB	1
500	500	0.01	50000	-	-	-	300	25	30	783.85MB	0.969
800	800	0.005	80000	-	-	-	400	30	50	2.02GB	0.973
1000	1000	0.005	100000	-	-	-	500	50	80	5.03GB	0.992

Table 1. Results on the synthetic dataset with ADGM and CURSOR.

*System runs out of memory.

eters for the experiments were set as:

First-order compatibility. The first-order compatibility matrix \mathbf{M} in Eq. (13) for all experiments was calculated as:

$$M_{i,j} = \exp(-\gamma_0 \|f_i - f_j\|_2) \quad (14)$$

where f_i and f_j are the normalized coordinates of i^{th} point in P_1 and j^{th} point in P_2 , respectively. The coordinates are normalized by subtracting the mean value of the coordinates in each set. γ_0 is the inverse of the mean value of all the distances from points in P_1 to the ones in P_2 . \mathbf{M} is then flattened column-wise as the first-order compatibility vector $\hat{\mathbf{m}}$.

Second-order compatibility matrix. The pairwise compatibility feature calculation of CURSOR followed [16], which found a balance between rotation and scale invariance. c columns, as \mathbf{C} , and another $3c^2$ entries of \mathbf{H} were randomly selected with a uniform distribution for the CUR decomposition. The soft-constraint assignment matrix was computed with the second-order PRL-based algorithm presented in Eq. (11).

Third-order compatibility tensor. Following [9], the same third-order compatibility feature calculation for the ANN-based and CURSOR methods was applied. The same t randomly selected hyperedges in the source graph were used for all the methods. The ANN-based and CURSOR methods generated the tensor with r_1 and r highest compatibilities from the target graph, respectively.

4.1. Large-Scale Random Synthetic Dataset

One thousand two-dimensional points, P , were sampled from a Gaussian distribution $\mathcal{N}(0, 1)$. Then, Gaussian deformation noise $\mathcal{N}(0, \sigma^2)$ was added to P as point set Q . During the experiments, n_1 points from P were selected as source graph \mathcal{G}_1 , and n_2 points containing the corresponding matching of \mathcal{G}_1 and outliers from Q were chosen as target graph \mathcal{G}_2 .

CURSOR was evaluated on synthetic datasets with increasing problem scales. Results of the ADGM algorithm based on ANN, whose accuracy was the highest among all the prior state-of-the-art hypergraph matching algorithms, were given. The parameter settings of ADGM, including t and r_1 , strictly followed the original work [19]. CURSOR can deal with large-scale scenarios and achieve privilege-matching accuracy with much less memory usage (Table 1). CURSOR was capable of solving 1000-vs-1000 matching problems with high accuracy, while ADGM failed to generate the tensor when $n_1 > 100$ under the same memory constraint. A more detailed memory footprint analysis and the potential bottleneck of CURSOR will be provided in the supplement.

4.2. Templates with Specific Shapes

To analyze the robustness of CURSOR to different shape deformations, middle-scale templates with specific shapes, which have been commonly used in previous works [14, 16, 29, 32] were evaluated. Four 2D templates were chosen: whale with 150 points, Chinese character with 105 points, UCF fish with 98 points, and tropical fish with 91 points. The target points were generated by rotation, scaling, with noise and outliers added. For CURSOR, $c = 100$, $k = 5$, $r = 25$ and for ANN-based methods, $r_1 = 300$. For both types of methods $t = 0.3n_1n_2$.

Different deformations were added on all the target points. For the rotation case, the source points were rotated with angle $\theta \in [-30^\circ, 30^\circ]$, then noise with $\sigma = 0.02\sigma_0$, where σ_0 is the standard deviation of all the source point coordinates, was added. For the scaling case, only the x -coordinate of the 2D point was scaled with the scaling factor $s = 1.1^\beta$, where $\beta \in [-5, 5]$, then noise with $\sigma = 0.02\sigma_0$ was added. To evaluate variant noise level, we vary noise with σ ranging from 0 to $0.1\sigma_0$ with step $0.01\sigma_0$. To add outliers, assume the mean value of the source points is μ_0 and randomly add n_l outliers from a Gaussian distribution

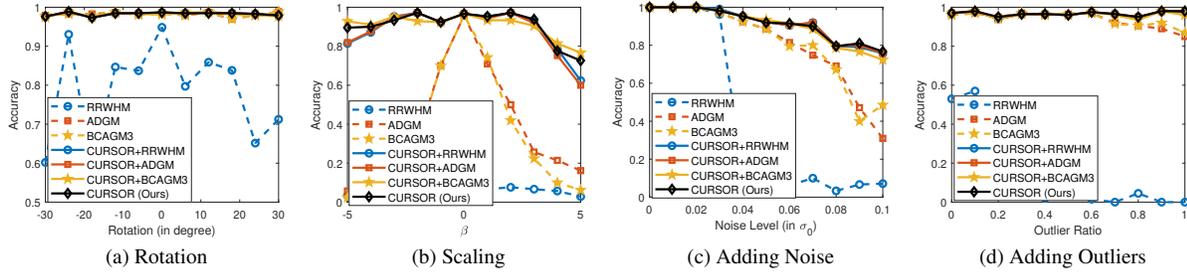


Figure 2. Matching result with deformation (a) rotation with angle $[-30^\circ, 30^\circ]$, (b) scale on x -coordinate with scale factor 1.1^β where $\beta \in [-5, 5]$, (c) adding noise with $\sigma/\sigma_0 = [0, 0.1]$, and (d) adding n_l outliers where the outlier ratio = n_l/n_1 .

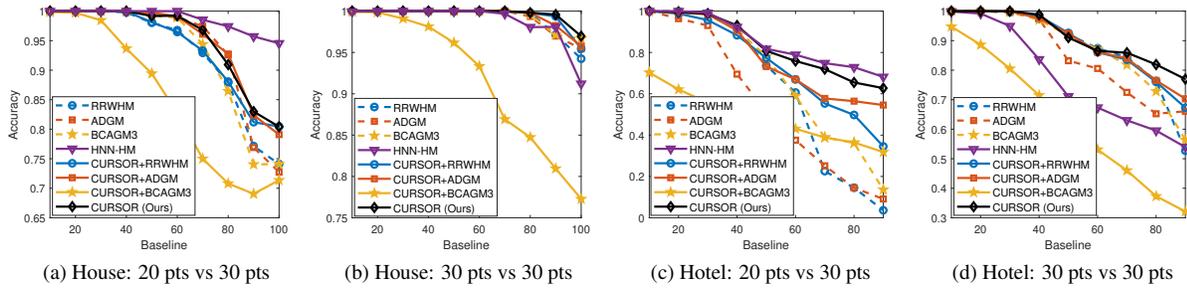


Figure 3. Comparison results on the House and Hotel dataset with various matching algorithms. The dashed curves represent the matching results on the compatibility tensors using ANN. The solid curves with the same color denote the matching accuracy on tensors generated by CURSOR with the same hypergraph matching algorithms.

$\mathcal{N}(\mu_0, \sigma_0^2)$, where the outlier ratio $n_l/n_1 \in [0, 1]$.

Due to the rotational invariance of hyperedge features, almost all the algorithms show an average accuracy of nearly 1 under a variant of rotation deformation whether ANN or CURSOR (Fig. 2). The curves of other three cases demonstrate that the matching algorithms with CURSOR outperform ANN. For example, with scaling, the hyperedge features were substantially damaged because the scaling process only scaled the x -coordinate of the target points. As the scale factor increased, the matching results of the ANN-based algorithms significantly decreased. In contrast, CURSOR generated compatibility tensors with higher robustness. The performance of the ANN matching algorithms was significantly improved and became competent after integrating the CURSOR tensor generation method. RRWHM performed much worse than other algorithms in all the cases but with the assistance of the compatibility tensor generated by CURSOR, its results became stable and showed comparable results to the other algorithms.

4.3. CMU House and Hotel Dataset

Previous works used the CMU House and Hotel datasets to evaluate the matching algorithms [9, 19, 20, 25, 30]. These datasets have 30 manually labeled feature points on a rotating 3D house or hotel model tracked over 111 and 101-frame image sequences, respectively. The experiment set-

tings of [19] were followed by matching all possible pairs with baseline (the separation between frames) varying from 10 to 100 in intervals of 10 for the House dataset and from 10 to 90 for the Hotel dataset. n_1 points were randomly selected in the first image to match all the points in the second for both datasets, where n_1 equaled 20 and 30 as two separate experiments. $t = n_1 n_2$ tuples were selected from the first image. For ANN-based methods, $r_1 = 200$ nearest neighbors were chosen for each hyperedge. For CURSOR, r was set to 25 and $c = 300$. For the learning-based HNN-HM, the training process of [25] on the House dataset was followed and the model was validated on both datasets.

The sequence-matching results are given in Fig. 3. All the learning-free-based matching algorithms combined with CURSOR surpassed the original ANN-based ones except for the BCAGM3 algorithm. The BCAGM3 algorithm with CURSOR frequently made descent errors during the experiment as its matching accuracy heavily decreased when the compatibility tensor was too sparse. CURSOR obtained comparable matching accuracy to the state-of-the-art learning-free algorithms when the outlier number was ten and achieved the best matching accuracy under the 30-vs-30 case on the House dataset. For the Hotel dataset, CURSOR achieved a much higher matching accuracy than ANN, but was less accurate than HNN-HM on 20-vs-30 problems as HNN-HM was specifically trained on the House Dataset.

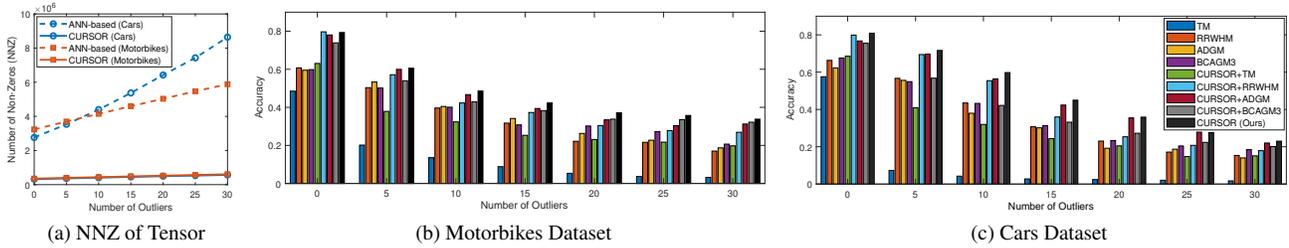


Figure 4. The Cars and Motorbikes datasets with CURSOR and state-of-the-art hypergraph matching algorithms. (a) The number of non-zero compatibilities with ANN-based methods and CURSOR. The matching accuracy on the (b) Motorbikes and (c) Cars datasets.

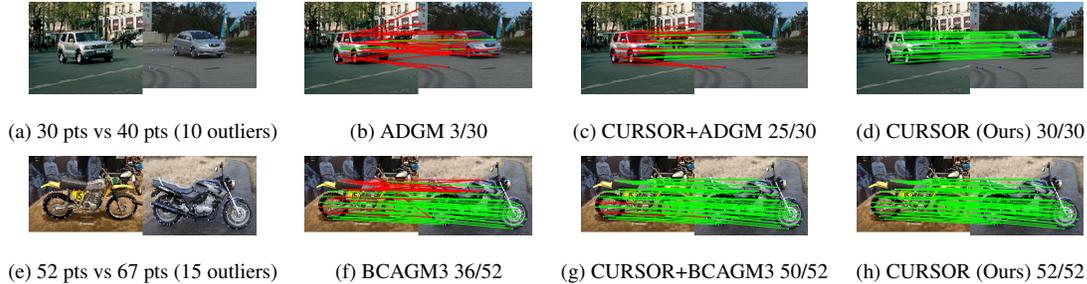


Figure 5. Car and Motorbike matching examples. Top row Car dataset, bottom row Motorbike dataset. Each example shows the matched results with the highest accuracy among trials. The green and red lines denote matches and mismatches, respectively.

It is noteworthy that CURSOR solved the matching problem with a unified model while HNN-HM must first train separating models in different datasets to achieve a better matching result. More detailed analysis will be provided in the supplement to show the effectiveness of CURSOR.

4.4. Car and Motorbike Dataset

The Car and Motorbikes dataset [22] consists of 30 real-life car image pairs and 20 motorbike image pairs, and was used in previous works to evaluate matching algorithms [19, 20, 30]. In this experiment, all inlier points in both images were kept and labeled outlier points were randomly chosen in the second image, with the number varying from 0 to 30 in step of 5. Every image pair in both datasets was matched. $t = n_1 n_2$ hyperedges were selected in the first image. For ANN-based methods, r_1 was set as $0.3n_1 n_2$. CURSOR selected $r = 50$ highest compatibilities in each tensor block. During experiments $k = 10$ and $c = 300$.

Figure 4a reported the average number of non-zero elements in the compatibility tensors generated by the two types of methods. The average accuracy with the ANN-based methods and CURSOR was shown in Fig. 4b and Fig. 4c. All algorithms integrated with CURSOR consistently improved their matching performance with a compatibility tensor more than ten times sparser than the ANN-based methods. In most cases, the default PRL-based matching algorithm achieved higher accuracy. Figure 5 shows matching examples. Combined with CURSOR,

other hypergraph matching algorithms showcased fewer mismatches. CURSOR, with the PRL-based algorithm, obtained the highest matching accuracy.

5. Conclusion

We propose CURSOR, a cascaded mixed-order hypergraph matching framework based on CUR decomposition for scalable graph matching. The framework contains a CUR-based second-order graph matching algorithm and a fiber-CUR-based tensor generation method, which significantly decreases the computational cost, and can be seamlessly integrated into existing state-of-the-art hypergraph matching algorithms to enhance their performance. A PRL-based hypergraph matching algorithm for sparse compatibility tensors is developed to accelerate the convergence. Experiment results demonstrated that CURSOR contributes to a higher matching accuracy with a sparser tensor, which has more potential utility in the big-data era. In future work, we plan to develop a more principled adaptive scheme to optimize the parameters of CURSOR so that it can perform better with fewer computations on larger-scale tasks.

Acknowledgment

This work is supported by the Hong Kong Innovation and Technology Commission (InnoHK Project CIMDA), the Hong Kong Research Grants Council (Project 11204821), and City University of Hong Kong (Project 9610034).

References

- [1] HanQin Cai, Keaton Hamm, Longxiu Huang, and Deanna Needell. Mode-wise tensor decompositions: Multi-dimensional generalizations of cur decompositions. *The Journal of Machine Learning Research*, 22(1):8321–8356, 2021. [3](#)
- [2] HanQin Cai, Keaton Hamm, Longxiu Huang, and Deanna Needell. Robust cur decomposition: Theory and imaging applications. *SIAM Journal on Imaging Sciences*, 14(4):1472–1503, 2021. [3](#)
- [3] Youcheng Cai, Lin Li, Dong Wang, Xinjie Li, and Xiaoping Liu. Htmatch: An efficient hybrid transformer based graph neural network for local feature matching. *Signal Processing*, 204:108859, 2023. [2](#)
- [4] Hongkai Chen, Zixin Luo, Jiahui Zhang, Lei Zhou, Xuyang Bai, Zeyu Hu, Chiew-Lan Tai, and Long Quan. Learning to match features with seeded graph matching network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6301–6310, 2021. [2](#)
- [5] Jiazhou Chen, Hong Peng, Guoqiang Han, Hongmin Cai, and Jiulun Cai. Hogmmnc: a higher order graph matching with multiple network constraints model for gene–drug regulatory modules identification. *Bioinformatics*, 35(4):602–610, 2019. [1](#)
- [6] Long Chen, Zhongying Zhao, and Hong Yan. A probabilistic relaxation labeling (prl) based method for c. elegans cell tracking in microscopic image sequences. *IEEE Journal of Selected Topics in Signal Processing*, 10(1):185–192, 2015. [5](#)
- [7] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part V 11*, pages 492–505. Springer, 2010. [1](#), [2](#), [4](#)
- [8] Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. *Advances in neural information processing systems*, 19, 2006. [1](#), [2](#)
- [9] Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 33(12):2383–2395, 2011. [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [11](#), [13](#)
- [10] François-Xavier Dupé, Rohit Yadav, Guillaume Auzias, and Sylvain Takerkart. Kernelized multi-graph matching. In *Asian Conference on Machine Learning*, pages 311–326. PMLR, 2023. [2](#)
- [11] Jiawei He, Zehao Huang, Naiyan Wang, and Zhaoxiang Zhang. Learnable graph matching: Incorporating graph partitioning with deep feature learning for multiple object tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5299–5309, 2021. [1](#)
- [12] Jian Hou, Huaqiang Yuan, and Marcello Pelillo. Game-theoretic hypergraph matching with density enhancement. *Pattern Recognition*, 133:109035, 2023. [1](#), [2](#)
- [13] Robert A Hummel and Steven W Zucker. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (3):267–287, 1983. [4](#), [11](#)
- [14] Bing Jian and Baba C Vemuri. Robust point set registration using gaussian mixture models. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1633–1645, 2010. [6](#)
- [15] Zheheng Jiang, Hossein Rahmani, Plamen Angelov, Sue Black, and Bryan M Williams. Graph-context attention networks for size-varied deep graph matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2343–2352, 2022. [2](#)
- [16] Sheheryar Khan, Mehmood Nawaz, Xu Guoxia, and Hong Yan. Image correspondence with cur decomposition-based graph completion and matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):3054–3067, 2019. [2](#), [6](#)
- [17] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. [5](#)
- [18] Eugene L Lawler. The quadratic assignment problem. *Management science*, 9(4):586–599, 1963. [1](#)
- [19] D Khuê Lê-Huu and Nikos Paragios. Alternating direction graph matching. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 4914–4922. IEEE, 2017. [1](#), [2](#), [5](#), [6](#), [7](#), [8](#), [13](#)
- [20] Jungmin Lee, Minsu Cho, and Kyoung Mu Lee. Hyper-graph matching via reweighted random walks. In *CVPR 2011*, pages 1633–1640. IEEE, 2011. [1](#), [2](#), [5](#), [7](#), [8](#), [13](#)
- [21] Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, pages 1482–1489. IEEE, 2005. [1](#), [2](#), [3](#), [4](#), [11](#)
- [22] Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *International journal of computer vision*, 96:28–45, 2012. [8](#)
- [23] Guchong Li, Gang Li, and You He. Distributed multiple resolvable group targets tracking based on hypergraph matching. *IEEE Sensors Journal*, 2023. [1](#)
- [24] Wuyang Li, Xinyu Liu, and Yixuan Yuan. Sigma: Semantic-complete graph matching for domain adaptive object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5291–5300, 2022. [1](#)
- [25] Xiaowei Liao, Yong Xu, and Haibin Ling. Hypergraph neural networks for hypergraph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1266–1275, 2021. [2](#), [5](#), [7](#)
- [26] Chuanju Liu, Dongmei Niu, Xinghai Yang, and Xiuyang Zhao. Graph matching based on feature and spatial location information. *The Visual Computer*, 39(2):711–722, 2023. [2](#)
- [27] Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009. [3](#)

- [28] Biao Min, Ray CC Cheung, and Hong Yan. A flexible and customizable architecture for the relaxation labeling algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(2):106–110, 2013. [5](#)
- [29] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010. [6](#)
- [30] Quynh Nguyen, Antoine Gautier, and Matthias Hein. A flexible tensor block coordinate ascent scheme for hypergraph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5270–5278, 2015. [1](#), [2](#), [5](#), [7](#), [8](#)
- [31] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020. [2](#)
- [32] Fu-Dong Wang, Nan Xue, Yipeng Zhang, Gui-Song Xia, and Marcello Pelillo. A functional representation for graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 42(11):2737–2754, 2019. [1](#), [2](#), [6](#)
- [33] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3056–3065, 2019. [2](#)
- [34] Tao Wang, He Liu, Yidong Li, Yi Jin, Xiaohui Hou, and Haibin Ling. Learning combinatorial solver for graph matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7568–7577, 2020. [2](#)
- [35] Meng-Yun Wu, Dao-Qing Dai, and Hong Yan. Prl-dock: Protein-ligand docking based on hydrogen bond matching and probabilistic relaxation labeling. *Proteins: Structure, Function, and Bioinformatics*, 80(9):2137–2153, 2012. [5](#)
- [36] Yanan Wu, He Liu, Songhe Feng, Yi Jin, Gengyu Lyu, and Zizhang Wu. Gm-mlic: graph matching based multi-label image classification. *arXiv preprint arXiv:2104.14762*, 2021. [1](#)
- [37] Miao Xu, Rong Jin, and Zhi-Hua Zhou. Cur algorithm for partially observed matrices. In *International Conference on Machine Learning*, pages 1412–1421. PMLR, 2015. [3](#)
- [38] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2684–2693, 2018. [2](#)
- [39] Zongwei Zhou, Junliang Xing, Mengdan Zhang, and Weiming Hu. Online multi-target tracking with tensor-based high-order graph matching. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 1809–1814. IEEE, 2018. [1](#)

CURSOR: Scalable Mixed-Order Hypergraph Matching with CUR Decomposition

Supplementary Material

6. PRL-based Method Derivation

Consider source graph $\mathcal{G}_1 = \{\mathcal{V}_1, \mathcal{E}_1\}$ and target graph $\mathcal{G}_2 = \{\mathcal{V}_2, \mathcal{E}_2\}$. Since the compatibility matrices/tensors are non-negative in graph matching, the original compatibility coefficient between $(i, l) \in \mathcal{E}_1$ and $(j, m) \in \mathcal{E}_2$, denoted as $r_{il}(j, m) \in [-1, 1]$ in [13], is updated to $R_{il}(j, m) = 0.5(r_{il}(j, m) + 1)$. The original PRL-based updating becomes

$$p_i^{(k+1)}(j) = \frac{p_i^{(k)}(j) \sum_{l=1}^{n_1} \sum_{m=1}^{n_2} R_{il}(j, m) p_l^{(k)}(m)}{\sum_m p_i^{(k)}(m) \sum_{l=1}^{n_1} \sum_{m=1}^{n_2} R_{il}(j, m) p_l^{(k)}(m)} \quad (15)$$

where $p_i(j)$ represents the probability that i^{th} node of \mathcal{V}_1 matches j^{th} node of \mathcal{V}_2 . As discussed in the main paper, $p_i^{(k)}(j)$ in the numerator plays the role of weight factor. Specifically, if we ignore the effect of $p_i^{(k)}(j)$, Eq. (15) becomes the power-iteration-like linear updating scheme commonly used in SM [21] and TM [9].

Assume the highest probability for $i_0 \in \mathcal{V}_1$ is $p_{i_0}(j_0)$. Hummel and Zucker proved that $p_{i_0}^{(k)}(j_0)$ consistently satisfies $\sum_{l=1}^{n_1} \sum_{m=1}^{n_2} R_{i_0 l}(j_0, m) p_l^{(k)}(m) \geq p_{i_0}^{(k)}(j_0)$ during the updating if \mathbf{R} , the matrix form of $R_{i_1 i_2}(j_1, j_2)$ for all (i_1, i_2) and (j_1, j_2) , is symmetric [13]. To further accelerate the convergence, the weighting factor is replaced from $p_i^{(k)}(j)$ to $\sum_{l=1}^{n_1} \sum_{m=1}^{n_2} R_{il}(j, m) p_l^{(k)}(m)$, which leads to

$$p_i^{(k+1)}(j) = \frac{[\sum_l \sum_m R_{il}(j, m) p_l^{(k)}(m)]^2}{\sum_j [\sum_l \sum_m R_{il}(j, m) p_l^{(k)}(m)]^2} \quad (16)$$

Define the compatibility coefficients as:

$$R_{i_1 i_2}(j_1, j_2) = \begin{cases} M_{i_1, j_1} & i_1 = i_2 \text{ and } j_1 = j_2 \\ \hat{H}_{i_1, j_1, i_2, j_2} & \text{otherwise} \end{cases} \quad (17)$$

where $M_{i,j}$ is the first-order compatibility between i^{th} node of \mathcal{V}_1 and j^{th} node of \mathcal{V}_2 . $\hat{H}_{i_1, j_1, i_2, j_2}$ denotes the compatibility between $(i_1, i_2) \in \mathcal{E}_1$ and $(j_1, j_2) \in \mathcal{E}_2$. The numerator of Eq. (16) becomes:

$$\hat{p}_i^{(k+1)}(j) = (M_{i,j} p_i^{(k)}(j) + \sum_l \sum_m \hat{H}_{i,l,j,m} p_l^{(k)}(m))^2 \quad (18)$$

$\hat{p}_i^{(k+1)}(j)$ is updated with the combination of first and second-order compatibilities. The main paper shows that for a consistent updating scheme, $\sum_l \sum_m \hat{H}_{i,l,j,m} p_l^{(k)}(m)^2 \geq 0.25 M_{i_0, j_0}^{-1}$. By replacing the probability set $\mathbf{p} = \{p_i(j)\}$ with the vector \mathbf{x} ,

the column-wise flattening of soft-constraint assignment matrix \mathbf{X} , Eq. (18) can be updated as

$$\hat{\mathbf{x}}^{(k+1)} = (\hat{\mathbf{m}} \odot \mathbf{x}^{(k)} + \mathbf{H}\mathbf{x}^{(k)})^2 \quad (19)$$

where \odot is the element-wise multiplication and the first-order compatibility vector $\hat{\mathbf{m}}$ is obtained by column-wise flattening $\hat{\mathbf{M}}$. The square calculation in Eq. (19) is also element-wise. \mathbf{H} is the second-order compatibility matrix. Since both \mathbf{H} and $\hat{\mathbf{m}}$ are dense, with the all-ones vector as $\mathbf{x}^{(0)}$, Eq. (19) converges consistently.

In our work, the updating scheme for PRL-based hypergraph matching is extended to

$$\hat{\mathbf{x}}^{(k+1)} = (\alpha \hat{\mathbf{m}} \odot \mathbf{x}^{(k)} + (1 - \alpha) \mathcal{H} \otimes_1 \mathbf{x}^{(k)} \otimes_2 \mathbf{x}^{(k)})^2 \quad (20)$$

where \otimes_l is the mode- l product of the tensor and vector and $\alpha \in [0, 1]$ is a balance weight between the first and third-order compatibilities. Since the third-order compatibility tensor \mathcal{H} is highly sparse, the high value of $\hat{\mathbf{x}}^{(k+1)}$ in Eq. (20) is concentrated if the sparse tensor is reliable. In our work, a reliable tensor means most ground truth hyperedge pair compatibilities are successfully selected in the tensor blocks.

7. Detailed Analysis in Sec. 4

Due to the space limitation of the main paper, we provide a more detailed experiment analysis based on the results of Sec. 4 to discuss the superiority and bottleneck of CURSOR.

7.1. Memory Footprint Analysis in Sec. 4.1

Table 2 shows the detailed memory footprint of the experiment result with CURSOR in Sec. 4.1 of the main paper. Theoretically, the CUR decomposition of the matrix \mathbf{H} , requires $O(cn_1 n_2)$ space complexity. The tensor \mathcal{H} , on the other hand, only needs $O(tr)$. For small-scale problems, the sparse tensor occupies most memory footprint with a small-size matrix. As the graph scale grows, with more columns selected from the compatibility matrices for higher matching accuracy, the main space occupation comes from \mathbf{H} , and the second-order CUR-based matching becomes the bottleneck for the graph matching problem. Although CURSOR can deal with larger-scale tasks compared to ANN, its capability to solve scalable problems is limited to the second-order matrix.

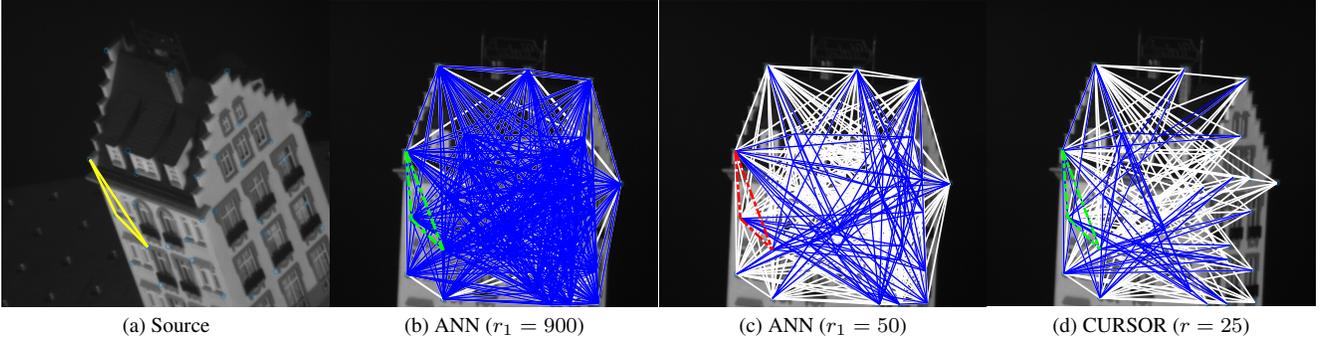


Figure 6. The detailed hyperedge correspondences between one sampled hyperedge from the source (yellow triangle in (a)) and target images with ANN (b) and (c) and CURSOR (d). The white triangles in the target images denote all the hyperedges compared with the source hyperedge, and the blue triangles are the hyperedges with the highest compatibilities (r_1 for ANN and r for CURSOR). The green dashed triangles represent the matched hyperedges, and the red one represents the mismatch.

Table 2. Detailed memory footprint of CURSOR in Tab. 1 of the main paper.

Problem	Parameter			Memory Footprint
	t	c	r	$\mathbf{H}/(\mathbf{H} + \mathcal{H})$
n_1 vs n_2				
30 vs 30	900	15	5	0.11MB/0.62MB
30 vs 50	1500	15	5	0.20MB/0.82MB
50 vs 50	2500	20	7	0.40MB/2.37MB
50 vs 100	5000	100	10	3.97MB/9.62MB
100 vs 100	10000	100	20	8.02MB/30.65MB
300 vs 300	30000	200	20	0.14GB/0.21GB
500 vs 500	50000	300	30	0.59GB/0.76GB
800 vs 800	80000	400	50	1.95GB/2.02GB
1000 vs 1000	100000	500	80	4.88GB/5.03GB

7.2. Visualization analysis in Sec. 4.3

We provide a more detailed analysis of the experiment results from Sec. 4.3 to demonstrate the superiority of CURSOR, as is shown in Fig. 6. Traditional ANN-based methods compute the compatibilities between the sampled source hyperedges (yellow triangle in Fig. 6a) and all the target ones (fully connected white triangles in Figs. 6b-6c). With the intermediate second-order result, CURSOR computes fewer compatibilities, as is shown in Fig. 6d. To find the corresponding hyperedge (green dashed triangles in Fig. 6), a large amount (the hyperparameter $r_1 = 900$ in Fig. 6b) of the highest compatibilities (blue triangles in Figs. 6b-6c) should be selected for ANN. If we decrease r_1 to 50, some correct ones will be missed (red triangle in Fig. 6c). CURSOR can effectively find the hyperedges with the 25 highest compatibilities (green triangle in Fig. 6d). Compared to the traditional tensor generation methods, CURSOR can effectively increase the matching performance with less computational complexity.

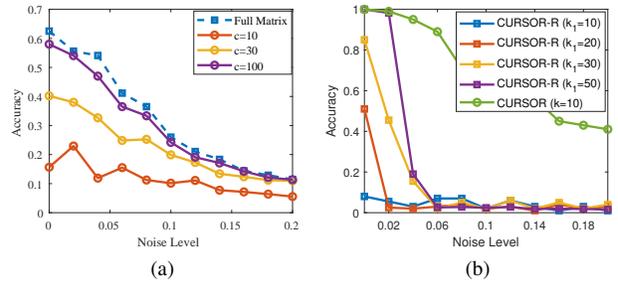


Figure 7. (a) CUR-based second-order graph matching accuracy with various c . (b) The average accuracy using CURSOR with CUR-based pairwise matching result and CURSOR with randomly sampled indices (CURSOR-R).

8. Ablation Studies

To further analyze the effectiveness of several design choices in CURSOR, ablation studies were conducted on the 100-vs-110 random synthetic dataset introduced in Sec. 4.1 of the main paper.

8.1. CUR-based Pairwise Matching

We first studied the effectiveness of the CUR-based second-order graph matching. Two experiments were designed to analyze the performance of the pairwise matching with the CUR decomposition of the second-order compatibility matrix \mathbf{H} and the rough intermediate matching result, respectively. The dataset's noise level σ varied in the range $[0, 0.2]$.

The CUR-based second-order graph matching was first evaluated on the dataset with various c , i.e., the number of randomly selected columns from \mathbf{H} . During the experiment, only the second-order graph-matching result was analyzed. We compared the proposed method with a PRL-based pairwise matching using the full compatibility ma-

Table 3. Average time consumption for second-order graph matching (in seconds)

Method	Time (s)	
	Computing \mathbf{H}	Matching
Full Matrix	6.311	0.491
$c = 10$	0.018	0.016
$c = 30$	0.031	0.016
$c = 100$	0.109	0.043

trix. The matching accuracy is shown in Fig. 7a. Due to the low-rank estimation, the CUR-based method decreased the matching performance. Specifically, as the columns were randomly chosen, the result was quite poor when c was relatively small. However, with less than 1% (100 out of 100×110) of columns selected, the CUR-based pairwise matching algorithm achieved a comparable result to the algorithm using the whole matrix. Table 3 reports the average time consumption of the compatibility matrix generation and graph matching. With only a few columns calculated, the CUR-based second-order algorithm effectively accelerated the matching process.

The effectiveness of the intermediate second-order matching result was further studied. As discussed in the main paper, the second-order matching result $\mathcal{P}^k = \{\mathcal{P}_1^k, \dots, \mathcal{P}_{n_1}^k\}$ consists k best-matching target nodes for each source node, where n_1 denotes the number of the source nodes. The hypergraph matching result with \mathcal{P}^k was compared to the result with k_1 randomly sampled indices in all three tensor modes (denoted as CURSOR-R). The parameter c and k from \mathcal{P}^k for CURSOR with the pairwise matching result was set as 100 and 10 respectively. For CURSOR-R, k_1 varied from 10 to 50. For both methods, the number of randomly selected hyperedges from the source hyperedges $t = 3000$, the number of highest compatibilities in each tensor block $r = 100$, and the balance factor of PRL-based algorithm α was set to 0.2. As shown in Fig. 7b, CURSOR-R had a lower matching accuracy even for $k_1 = 50$. Assuming the number of nodes is n_2 in the target graph, theoretically, only 3 out of $3n_2^2$ fibers for each tensor block, i.e., key fibers in our work, contain the entry of the ground truth hyperedge pair. Due to the random sampling, the probability of selecting the key fiber is $(k_1/n_2)^2$, around 20.7% when $k_1 = 50$ and $n_2 = 110$. Therefore, the sparse compatibility tensor generated by CURSOR-R was highly unreliable. CURSOR with the CUR-based pairwise matching result selected the key fiber in each tensor block with a high probability, effectively increasing the final accuracy.

8.2. CURSOR vs ANN-based Tensor Generation

Experiment results in Sec. 4 of the main paper have already shown that the proposed PRL-based algorithm with

CURSOR achieved higher matching accuracy than other algorithms in most cases. One may wonder how the ANN-based tensor generation performs with the same hypergraph matching algorithm. In this experiment, we further compare CURSOR with the ANN-based tensor generation method applying PRL-based algorithm on the 100-vs-110 random synthetic dataset. During experiment, $c = 100$, $k = 10$ and $r = 100$ for CURSOR. For ANN-based tensor generation, r_1 was set as 100 for the same tensor density. For both methods, $t = 3000$ and $\alpha = 0.2$. The stopping criterion of the PRL-based algorithm was set as $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2 \leq 10^{-8}$ and the maximum number of iteration was 100.

The average matching accuracy is reported in Fig. 8a. The ANN method shows an unstable performance with high σ since the compatibility tensor was too sparse. When α was low, since the PRL-based algorithm focused on the third-order compatibilities, the ANN method did not generate a reliable compatibility tensor. To make the algorithm focus more on first-order compatibilities, α was further increased to 0.8 for the ANN case, which significantly improved the accuracy. The average number of iterations to converge is shown in Fig. 8b. When $\sigma > 0.02$, the ANN method did not converge within 100 iterations. With the same tensor sparsity, CURSOR successfully converged in all the cases. The decay of $\mathbf{x}^{(k)}$ per iteration for the ANN method and CURSOR with $\sigma = 0.1$ was further analyzed, as shown in Fig. 8c. The ANN method did not converge due to few ground truth hyperedge compatibilities selected from the whole tensor. However, CURSOR chose the non-zero compatibilities from a smaller reliable searching region. Therefore, it is capable of converging fast with the same tensor sparsity.

9. Parameter Sensitivity Analysis

Experiments below are provided to investigate how the hyperparameters in CURSOR affect the final results. Since the hyperparameter t , the number of randomly selected hyperedges, was thoroughly studied in previous works [9, 19, 20], we do not redundantly analyze it here. The method was evaluated on the random 100-vs-110 synthetic dataset introduced in Sec. 4.1 of the main paper as well.

9.1. Parameter in Second-order Graph Matching

The sensitivity of parameter c to the whole framework was first analyzed. During the experiment, the whole compatibility matrix was calculated directly with noise level $\sigma = 0.02$. CUR decomposition was evaluated on the matrix with various c for the pairwise graph matching. To analyze the influence of c on the second-order matching result, we define hit rate, calculated as $\sum_{i=1}^{n_1} \delta_i / n_1$, where

$$\delta_i = \begin{cases} 1 & \text{The true match } j \in \mathcal{P}_i^k \\ 0 & \text{Otherwise} \end{cases} \quad (21)$$

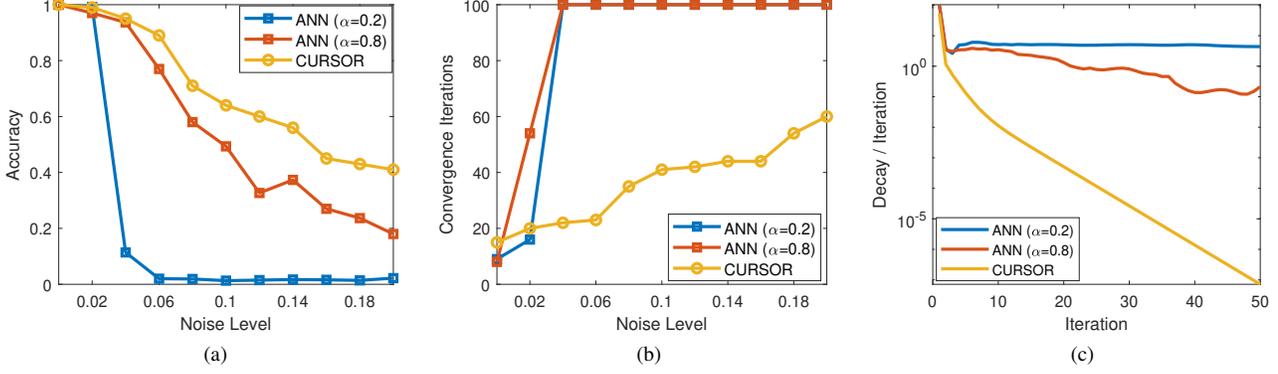


Figure 8. Results on 100-vs-110 synthetic dataset comparing CURSOR with ANN, $\alpha = 0.2$ for CURSOR. (a) The average matching accuracy using CURSOR and ANN with different α . (b) The average iterations for convergence using CURSOR and ANN with different α . (c) The decay $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2$ per iteration with different α using CURSOR and ANN. The noise level $\sigma = 0.1$.

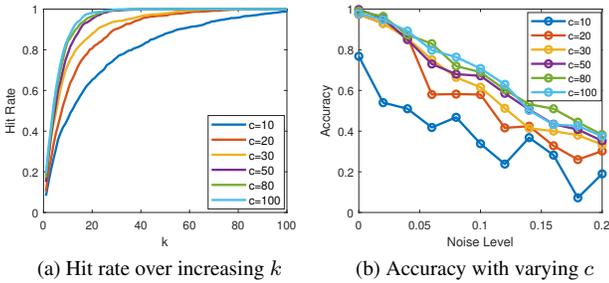


Figure 9. (a) The average hit rate over increasing k on a 100-vs-110 synthetic dataset with $\sigma = 0.02$ from the CUR-based second-order graph matching results. (b) The average accuracy on the 100-vs-110 synthetic dataset with $\sigma \in [0, 0.2]$ and $c \in [10, 100]$.

The average hit rate was computed over the increasing number of selected highest compatibilities k , as shown in Fig. 9a. Each curve represents the hit rate with c during CUR decomposition. To achieve the same hit rate and set k as small as possible, theoretically, the number of selected columns needed to be as large as possible. However, when increasing c from 50 to 100, the gain on hit rate is minor, with a relatively small k to reach a promising hit rate like 0.9. Therefore, to balance the time consumption and the matching performance, a relatively small proportion of columns is sufficient for the rough pairwise matching result.

The influence of c on the final matching result was further analyzed. During the experiment, we set $k = 10$, $r = 100$, and $\alpha = 0.2$. The noise level of the dataset was assigned as $\sigma \in [0, 0.2]$. The result is reported in Fig. 9b. Since the columns of the compatibility matrices were randomly selected, the matching result was unstable when c was less than 20. The performance gradually saturated with over 50 columns (around 0.5% of the total columns). The experiment results in Fig. 9a show that 50-100 columns can

already provide a reliable second-order matching result for the following higher-order process, effectively decreasing the computation cost.

9.2. Parameters During Tensor Generation

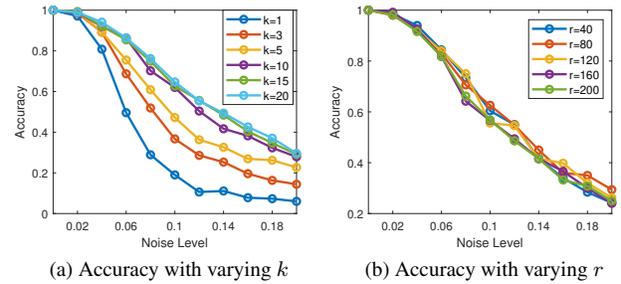


Figure 10. Results on 100-vs-110 synthetic dataset with $\sigma \in [0, 0.2]$ using CURSOR. (a) The matching accuracy with increasing k . (b) The matching accuracy with increasing r .

The sensitivity of k and r to the final matching accuracy was further studied. During the experiment, c was set as 100 and $\alpha = 0.2$.

To figure out the influence of k , the parameter r was set as 100, and k varied from 1 to 20. The result is shown in Fig. 10a. It is obvious that a higher k can achieve higher robustness for target points with high noise impact. However, with a polynomial $O(tk^2n_2)$ computation cost for tensor generation, the matching performance gradually reached its peak. For instance, the matching accuracy increased more than 35% from $k = 1$ to $k = 5$ when $\sigma = 0.08$, but less than 2% from $k = 15$ to $k = 20$. For each tensor block, if the entry of the ground truth paired hyperedge compatibility was included in r non-zero elements, the corresponding nodes would be matched with a high probability. Therefore, an appropriate k needs to be set to balance the computation

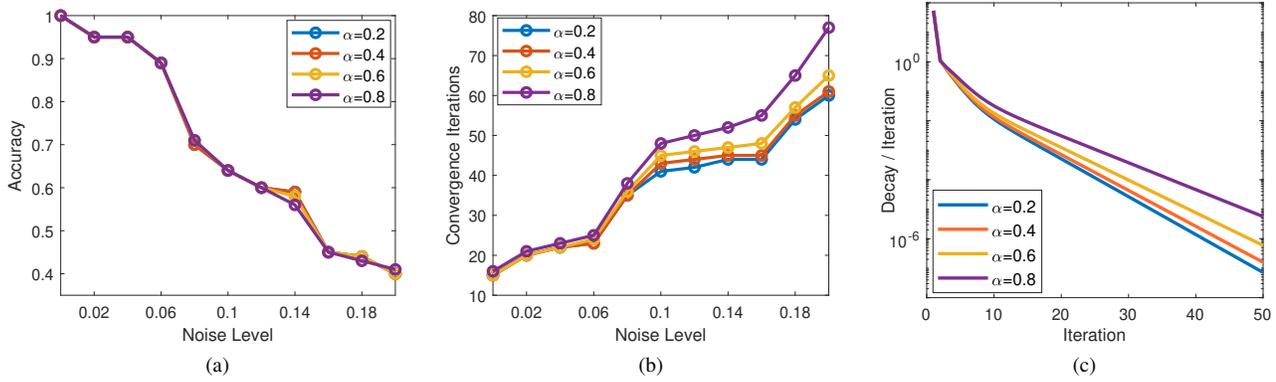


Figure 11. Results on 100-vs-110 synthetic dataset with PRL-based matching algorithm. (a) The average matching accuracy using CURSOR with different α . (b) The average iterations for convergence using CURSOR with different α . (c) The decay $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2$ per iteration with different α settings using CURSOR. The noise level $\sigma = 0.1$.

cost and the performance in practical use.

The effect of r was further analyzed by setting k as 10 and varying r from 20 to 200, as shown in Fig. 10b. Unlike the results in Fig. 10a, CURSOR with the highest r performed the worst. The reason may be that when the ground truth hyperedge pair is already included in the non-zero compatibilities, more redundant compatibilities can cause lower matching performance.

9.3. Parameters of PRL-based Matching Algorithm

To study the sensitivity of α , the matching accuracy and convergence speed using CURSOR were further analyzed. During the experiment, $c = 100$, $k = 10$, and r was set as 100. The stopping criteria of the PRL-based algorithm was assigned as $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2 \leq 10^{-8}$. The matching accuracy with various α is shown in Fig. 11a. With a reliable compatibility tensor, the PRL-based matching algorithm achieved almost the same performance regardless of α . We further analyzed their convergence speeds, as shown in Figs. 11b and 11c. As the noise level increased, more iterations were required to satisfy the stopping criteria, and a lower α achieved faster convergence. As discussed in the previous sections, the parameter α is a balanced factor between first and third-order compatibilities. Although the method converged faster with $\alpha = 0$, the first-order compatibilities stabilized the matching process and increased the matching performance under some extreme circumstances.