Supplementary Material of Joint t-SNE for Comparable Projections of Multiple High-Dimensional Datasets

Category: Research

Paper Type: algorithm/technique

1 Introduction

We provide technical details and more experimental results that are not presented in the paper due to the page limit. This supplementary material is organized as follows: Section 2 gives the pseudo-code of both the original GUISE algorithm and our modified version. Section 3 provides the parameter settings for all our experiments. We also compare the original graphlet enumeration algorithm with our random walk algorithm in terms of time cost and accuracy in Section 4. Section 5 and 6 present projection results of the 10-Gaussian dataset and the VGG dataset, respectively. The detailed quantitative measurement for the four datasets is shown in Section 7. We discuss the generalization of our method to other multidimensional projection schemes in Section 8.

2 Algorithm Details

In this section, we provide algorithm details of Joint t-SNE , including computation of GFD-based feature vector, original and modified version of GUISE, computation of point and edge similarity.

Algorithm 1 Compute the normalized GFD around one node

Require: GL is a set of graphlets contained in a graph, v is the target vertex to compute the feature vector
Ensure: A vector of size 29 that represents the normalized frequencies of 29 graphlets which contain node v

```
1: procedure NODE_FEATURE(GL, v)
2:     nodeGFD ← ZEROS(29)
3:     for gl ∈ GRAPHLETS_WITH_NODE(GL, v) do
4:         type ← GET_GRAPHLET_TYPE(gl)
5:         nodeGFD[type] ← nodeGFD[type] + 1
6:     end for
7:     NORMALIZE(nodeGFD)
8:     return nodeGFD
9: end procedure
```

3 Parameter settings

For all techniques, we adopted the same optimization procedure as the original t-SNE paper except for the number of iterations $T = 2,000$. A common hyperparameter *perplexity* was chosen depending on the dataset: 70 for the 10-Gaussian datasets, 40 for the 5-Gaussian datasets, 40 for the MNIST dataset, and 50 for the VGG dataset.

k is set to 3 for all dataset.As for γ or λ, they are set to 0.1 in the 5-Gaussian datasets, the 10-Gaussian datasets and the MNIST dataset, and exclusively 0.01 for the VGG dataset.

4 Performance

We evaluate the effectiveness and efficiency of counting graphlets using a random walk-based method. First, we compute the average L1 loss between feature vectors calculated by random walk and those by enumeration in Table 1. Second, we compare the average time taken to compute feature vectors with different graph sizes between our method

Algorithm 2 Accelerating graphlet enumeration using random walk

Require: G is a graph

Ensure: GL is a set of uniformly sampled graphlets in G

procedure GUISE_ON_DISCONNECTED_GRAPH(G)
  $GL \leftarrow \emptyset$
  $ConnCpnts \leftarrow$ GET_CONNECTED_COMPONENTS(G)
  for $cpnt \in ConnCpnts$ **do**
    $SCount \leftarrow cpnt.size() \times 1000$
    $GL.append(\text{GUISE}(cpnt, SCount))$
  end for
  return GL
end procedure

Algorithm 3 Compute point similarity

Require: Two points $v_i^0 \in V(G_0)$ and $v_i^1 \in V(G_1)$, GL_0 and GL_1 are two sets of graphlets of graph G_0 and G_1 respectively, k is the number of nearest neighbors considered

Ensure: The point similarity between v_i^0 and v_i^1

procedure POINT_SIMILARITY($v_i^0, v_i^1, G_0, G_1, GL_0, GL_1, k$)
  $fv_i^0 \leftarrow$ NODE_FEATURE(GL_0, v_i^0)
  $fv_i^1 \leftarrow$ NODE_FEATURE(GL_1, v_i^1)
  $rate \leftarrow \|kNN(G_0, v_i^0, k) \cap kNN(G_1, v_i^1, k)\|/k$
  return $rate \cdot < fv_i^0, fv_i^1 >$ ▷ $< \cdot, \cdot >$ is the cosine similarity
end procedure

Algorithm 4 Compute edge similarity

Require: Two edges $e_{ij}^0 \in E(G_0)$ and $e_{ij}^1 \in E(G_1)$, GL_0 and GL_1 are two sets of graphlets of graph G_0 and G_1 respectively, k is the number of nearest neighbors considered

Ensure: The edge similarity between e_{ij}^0 and e_{ij}^1

procedure EDGE_SIMILARITY($e_{ij}^0, e_{ij}^1, G_0, G_1, GL_0, GL_1, k$)
  $(v_i^0, v_j^0) \leftarrow e_{ij}^0$
  $(v_i^1, v_j^1) \leftarrow e_{ij}^1$
  $s_i \leftarrow$ POINT_SIMILARITY($v_i^0, v_i^1, G_0, G_1, GL_0, GL_1, k$)
  $s_j \leftarrow$ POINT_SIMILARITY($v_j^0, v_j^1, G_0, G_1, GL_0, GL_1, k$)
  return $s_i \cdot s_j$
end procedure

Algorithm 5 Compute common edge similarities

Require: G_0 and G_1 are two graphs, k is the number of nearest neighbors considered
Ensure: Similarities of common edges between G_0 and G_1

procedure COMMON_EDGE_SIMILARITIES(G_0, G_1, k)
  $GL_0 \leftarrow$ GUISE_ON_DISCONNECTED_GRAPH(G_0)
  $GL_1 \leftarrow$ GUISE_ON_DISCONNECTED_GRAPH(G_1)
  $(V_0, E_0) \leftarrow G_0$
  $(V_1, E_1) \leftarrow G_1$
  $E_{com} \leftarrow E_0 \cap E_1$
  $sims \leftarrow \emptyset$
  for $(e^0_{ij}, e^1_{ij}) \in E_{com}$ **do**
    $sims[e^0_{ij}, e^1_{ij}] \leftarrow$ EDGE_SIMILARITY($e^0_{ij}, e^1_{ij}, G_0, G_1, GL_0, GL_1, k$)
  end for
  return *sims*
end procedure

Algorithm 6 Joint t-SNE

Require: X_0 and X_1 are datasets from two adjacent time frames, Y_0 is the projection of X_0, $Perp$ is the perplexity in the t-SNE loss function, k is the parameter for building kNN graph, and γ is the weight for vector constraint
Ensure: The projection of X_1

procedure JOINT t-SNE(X_0, X_1, Y_0, $Perp$, $k = 3$, $\gamma = 0.1$)
  $G_0 \leftarrow$ BUILD_KNN_GRAPH(X_0, k)
  $G_1 \leftarrow$ BUILD_KNN_GRAPH(X_1, k)
  $S_e \leftarrow$ COMMON_EDGE_SIMILARITIES(G_0, G_1, k)
  $Y_1 \leftarrow \arg\min$ LOSS($Y_0, X_1, Perp, S_e, \gamma$)
  return Y_1
end procedure

Algorithm 7 Uniform Sampling Algorithm

procedure GUISE($G, SCount$)
  $graphlets \leftarrow []$
  $g_x \leftarrow$ GET_A_INITIAL_GRAPHLET(G)
  $d_{g_x} \leftarrow$ POPULATE_NEIGHBORHOOD(g_x)
  $sampled \leftarrow 0$
  while True **do**
    choose a neighbor g_y uniformly from all possible neighbors
    $d_{g_y} \leftarrow populate\_neighborhood(g_y)$
    $acceptance\_probability \leftarrow \mathbf{min}(\frac{|d_{g_x}|}{|d_{g_y}|}, 1)$
    if $uniform(0,1) \leq acceptance\_probability$ **then**
      $g_x \leftarrow g_y$
      $d_{g_x} \leftarrow d_{g_y}$
    end if
    $sampled \leftarrow sampled + 1$
    $graphlets.append(g_x)$
    if $sampled > SCount$ **then return** $graphlets$
    end if
  end while
end procedure
procedure POPULATE_NEIGHBORHOOD(g_x)
  $neighbor\_list \leftarrow$ generate all potential neighboring graphlets
  return $neighbor\_list$
end procedure

and simple enumeration in Figure 1. It shows that the random walk-based method is much faster than brute-force enumeration with a reasonable loss which is less than 0.5.

We also report the actual running time of Joint t-SNE as well as other methods in Table 3.

Table 1: error caused by random walk

# of Edges	L1 error
100	0.0103226
200	0.349447
300	0.499137
400	0.532747
500	0.439423
600	0.424821
700	0.424222
800	0.462328
900	0.434751
1,000	0.396056

5 Projecting 10-Gaussian dataset

This synthetic dataset is generated as described in section 5.1 in the paper. The projection results are shown in Figure 2.

6 Projecting the Activation of VGG-16 Network

Dataset We use the same VGG dataset as in the paper but apply t-SNE , Equal-initialization t-SNE , and Dynamic

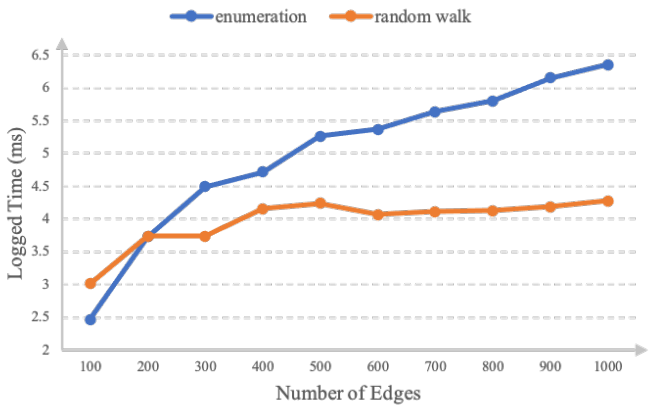

Fig. 1: Time cost of graphlet enumeration vs. random walk

Table 2: Quantitative measurement for projection fidelity

Methods / Datasets	*k*NN preservation				KL divergence			
	t-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**
5-Gaussian	0.30	0.30	0.32	**0.34**	1.00	1.00	1.05	**1.03**
10-Gaussian	0.19	0.19	0.16	**0.23**	1.62	1.62	1.69	**1.69**
MNIST	0.26	0.26	0.21	**0.24**	1.00	1.00	1.16	**1.05**
VGG	0.57	0.56	0.48	**0.55**	0.60	0.60	1.01	**0.65**

Table 3: Comparison of time performance between three methods

Time(s) / Datasets	t-SNE	DT	JT
5-Gaussian	74.67	142.35	187.10
10-Gaussian	924.87	1736.14	1506.24
MNIST	24.74	73.83	47.34
VGG	175.50	269.02	391.33

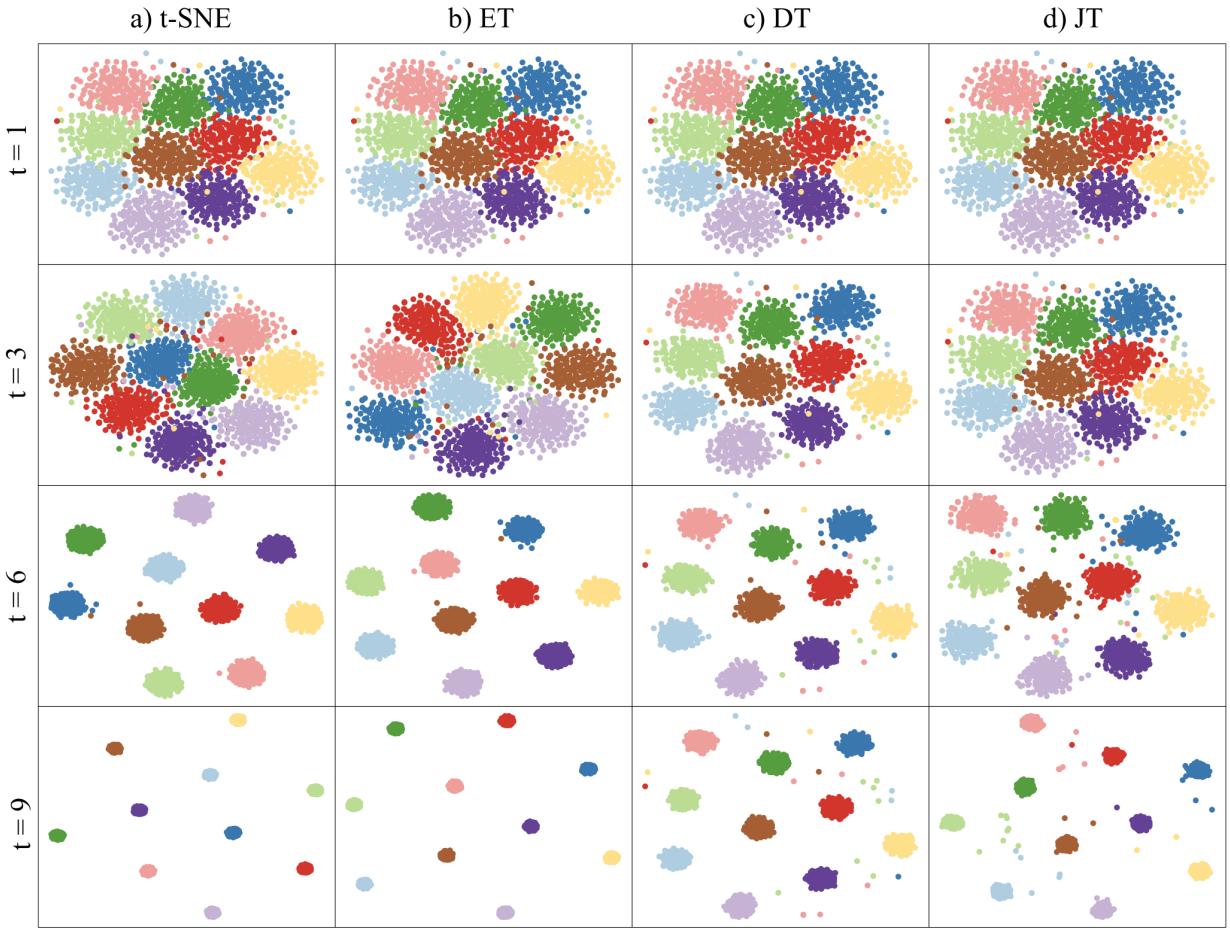

Fig. 2: Comparison of the 10-Gaussian dataset projection of four different t-SNE techniques. Both t-SNE and Equal-initialized t-SNE failed in terms of maintaining visual consistency. For Dynamic t-SNE , the results seem more stable than ours since Joint t-SNE focuses on preserving relative positions of points within a cluster. However, our result can achieve smaller LCE than Dynamic t-SNE (Tables 10, 11, 12).

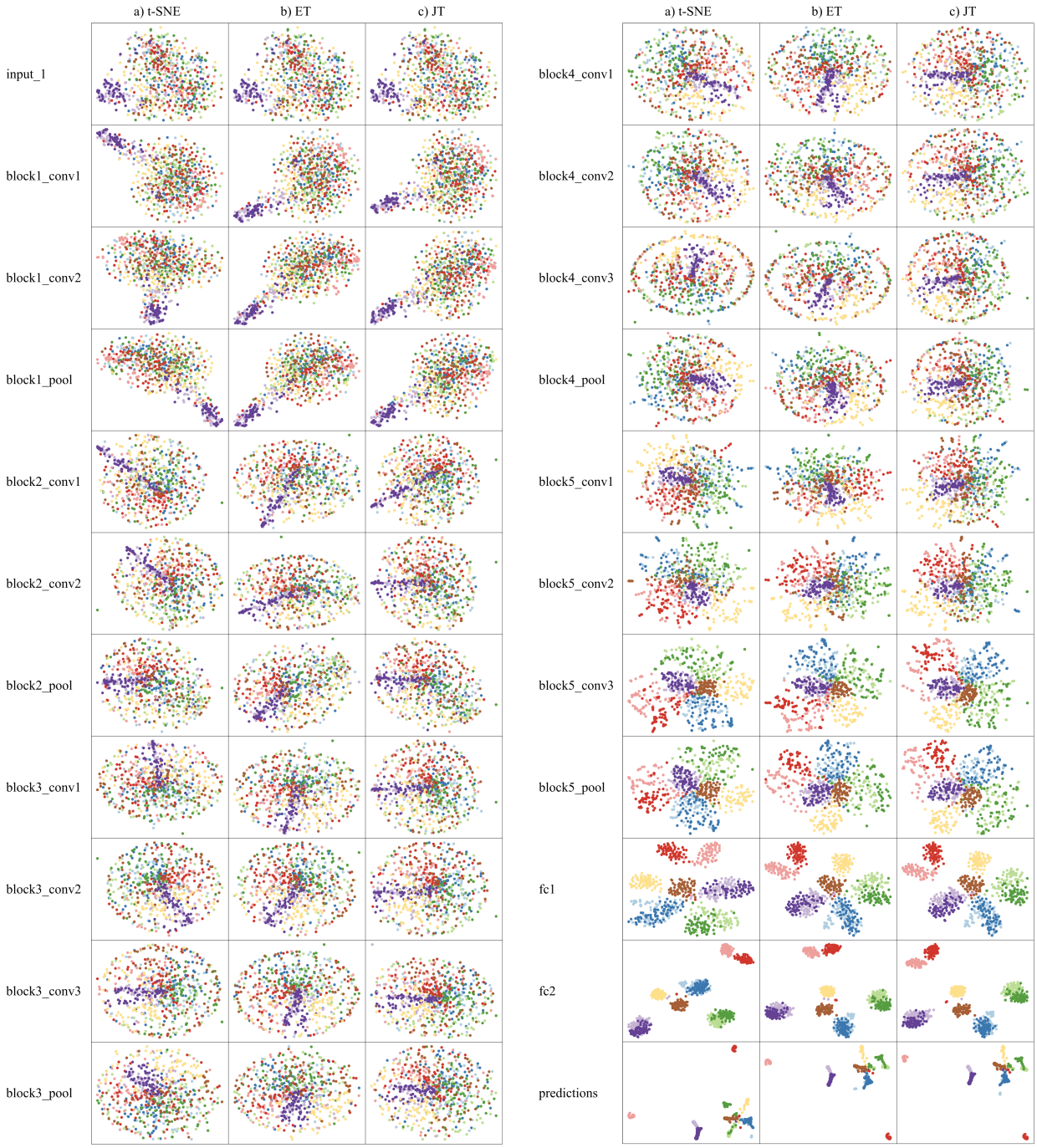

Fig. 3: The layer activation of 700 images of 10 classes in the VGG-16 network with different techniques excluding Dynamic *t*-SNE .

t-SNE on the activation of the complete 22 layers(from *input_1* to *predictions*). Note that we do not include the result of Dynamic t-SNE since the optimization process of Dynamic t-SNE always collapsed.

Projection Parameters Different with the default hyperparameters, we use $\gamma = \lambda = 0.01$, $perplexity = 50$, $k = 3$.

7 Quantitative Analysis

We report the projection fidelity and LCE of each dataset here. As we can see, in most cases, Joint t-SNE can achieve better performance than other techniques.

Table 4: Projection fidelity for MNIST dataset

	kNN preservation				KL divergence			
	t-SNE	ET	DT	**JT**	t-SNE	ET	DT	**JT**
t=0	0.59	0.59	0.49	**0.59**	0.52	0.52	0.98	**0.52**
t=1	0.54	0.54	0.47	**0.52**	0.67	0.68	1.03	**0.78**

Table 5: Projection fidelity for 5-Gaussian dataset

	kNN preservation				KL divergence			
	t-SNE	ET	DT	**JT**	t-SNE	ET	DT	**JT**
t=0	0.32	0.32	0.32	**0.32**	1.37	1.37	1.37	**1.37**
t=1	0.29	0.30	0.33	**0.34**	0.94	0.94	0.98	**0.97**
t=2	0.28	0.29	0.33	**0.36**	0.77	0.77	0.83	**0.82**
t=3	0.29	0.30	0.29	**0.33**	0.92	0.92	1.00	**0.96**

Table 6: Projection fidelity for 10-Gaussian dataset

	kNN preservation				KL divergence			
	t-SNE	ET	DT	**JT**	t-SNE	ET	DT	**JT**
t=1	0.21	0.21	0.21	**0.21**	2.51	2.51	2.51	**2.51**
t=3	0.30	0.30	0.21	**0.25**	1.97	1.97	2.00	**1.99**
t=6	0.16	0.16	0.14	**0.24**	1.19	1.19	1.27	**1.31**
t=9	0.09	0.09	0.09	**0.22**	0.80	0.80	0.96	**0.94**

8 Generation to other algorithm

We take MDS as an example to illustrate how we adapt our concepts to other projection algorithms. In general, given the two datasets, X_0 and X_1, we project X_0 using conventional MDS as follows:

Table 7: Projection fidelity for VGG dataset

	kNN preservation				KL divergence			
	t-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**
Block5 conv	0.07	0.07	0.07	**0.07**	1.86	1.86	1.86	**1.86**
Block5 pool	0.19	0.19	0.17	**0.16**	1.36	1.37	1.48	**1.44**
fc1	0.33	0.34	0.25	**0.31**	0.69	0.68	0.85	**0.69**
fc2	0.38	0.38	0.27	**0.36**	0.44	0.44	0.73	**0.47**

Table 8: Local Coherence Error for MNIST dataset

cluster 0				
	t-SNE	ET	DT	**JT**
t=1	15,951.53	6,710.96	5.80	**190.42**

Table 9: Local Coherence Error for 5-Gaussian dataset

	cluster 0				cluster 4			
	t-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**
t=1	2,191.86	2,191.86	71.84	**59.45**	2,171.73	1,689.15	34.76	**0.99**
t=2	2,071.51	1,855.28	24.54	**1.02**	2,156.03	1,372.43	40.93	**2.94**
t=3	1,666.01	609.04	3.74	**6.71**	1,577.11	1,035.27	5.45	**10.81**

Table 10: Local Coherence Error for 10-Gaussian dataset cluster 0-2

	cluster 0				cluster 1				cluster 2			
	t-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**
t=3	9,356.08	8,908.52	104.96	**6.55**	8,225.19	12,243.68	145.37	**6.93**	10,204.24	9,268.73	216.66	**5.78**
t=6	8,249.52	8,418.65	377.32	**149.21**	7,501.15	11,270.48	631.11	**83.35**	16,751.15	8,862.76	653.47	**213.60**
t=9	963.51	1,721.81	129.69	**34.90**	1,718.50	1,017.45	180.75	**53.23**	1,758.04	1,173.45	178.06	**77.22**

Table 11: Local Coherence Error for 10-Gaussian dataset cluster 3-5

	cluster 3				cluster 4				cluster 5			
	t-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**
t=3	8,895.99	8,717.15	116.07	**7.59**	10,695.90	7,968.67	154.24	**8.74**	6,574.76	7,031.94	74.72	**4.75**
t=6	14,775.11	17,997.41	455.99	**68.23**	9,995.13	7,979.74	436.99	**110.76**	15,253.66	9,794.78	288.54	**156.81**
t=9	1,081.16	1,609.94	138.15	**43.10**	953.66	1,610.65	147.98	**52.64**	1,184.60	1,626.42	140.47	**28.30**

Table 12: Local Coherence Error for 10-Gaussian dataset cluster 6-9

	cluster 6				cluster 7				cluster 8				cluster9			
	t-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**	*t*-SNE	ET	DT	**JT**
t=3	7,898.10	3,366.97	104.62	**13.58**	8,168.68	8,088.39	282.02	**6.57**	5,600.47	9,199.76	156.18	**8.03**	7,636.47	6,230.19	60.93	**4.37**
t=6	13,038.54	8,114.85	405.79	**113.20**	13,788.25	10,576.47	448.83	**84.80**	7,743.68	11,524.66	407.80	**92.80**	15,501.99	16,298.55	425.48	**178.98**
t=9	1,926.53	571.13	154.38	**54.23**	1,780.14	439.04	158.13	**31.51**	1,426.32	1,041.62	136.56	**58.07**	2,327.76	1,986.31	185.13	**73.20**

$$\arg\min_{Y_0} C = \frac{1}{N^2} \sum_i \sum_j \left(\left\| x_i^0 - x_j^0 \right\| - \left\| y_i^0 - y_j^0 \right\| \right)^2 \tag{1}$$

For X_1, we introduce vector constraints to the objective function as follows:

$$\arg\min_{Y_1} C = \frac{1}{N^2} \sum_i \sum_j \left(\left\| x_i^1 - x_j^1 \right\| - \left\| y_i^1 - y_j^1 \right\| \right)^2 \tag{2}$$

$$+ \frac{\gamma}{M} \sum_{i \neq j} S_{e_{ij}} \cdot \left\| (y_i^0 - y_j^0) - (y_i^1 - y_j^1) \right\|^2 \tag{3}$$

where $S_{e_{ij}}$ is the similarities of common edges computed based on GFD, M is the number of those common edges, γ is the weight for vector constraints set by users.