Test

In [168…
```
import torch
import matplotlib.pyplot as plt

import sys
sys.path.append('/home/jiahang/DiGress')

from src.datasets.protein_dataset import ProteinGoldenDataset,
ProteinInferDataset
from src.utils import get_e_mask, to_dense, to_dense_adj, slope_sigmoid
import numpy as np
```

In [124…
```
path_Gs_G0 = '/home/jiahang/DiGress/chain_results/2024-05-01/16-00-27-
protein-2/protein-2/test/Gs_G0.pkl'
path_Gs_Gt = '/home/jiahang/DiGress/chain_results/2024-05-01/16-00-27-
protein-2/protein-2/test/Gs_Gt.pkl'
path_Gs_Gt_infer = '/home/jiahang/DiGress/chain_results/2024-05-01/16-00-
27-protein-2_infer/protein-2_infer/infer/Gs_Gt.pkl'
```

In [26]:
```
import pickle
with open(path_Gs_G0, 'rb') as f:
    Gs_G0 = pickle.load(f)
with open(path_Gs_Gt, 'rb') as f:
    Gs_Gt = pickle.load(f)
```

In [125…
```
with open(path_Gs_Gt_infer, 'rb') as f:
    Gs_Gt_infer = pickle.load(f)
```

In [126…
```
def process_dist(dist):
    step_dist_list = []
    for step_idx in range(dist[0].shape[0]):
        _step_dist_list = []
        for batch in dist:
            _step_dist_list.append(batch[step_idx])
        _step_dist_list = torch.concat(_step_dist_list)
        step_dist_list.append(_step_dist_list)
    step_dist = torch.stack(step_dist_list)
    return step_dist.numpy()

Gs_G0_step_dist = process_dist(Gs_G0)
Gs_Gt_step_dist = process_dist(Gs_Gt)
```

In [323…
```
Gs_Gt_infer[0].shape
```

Out[323…
```
torch.Size([19, 51674])
```

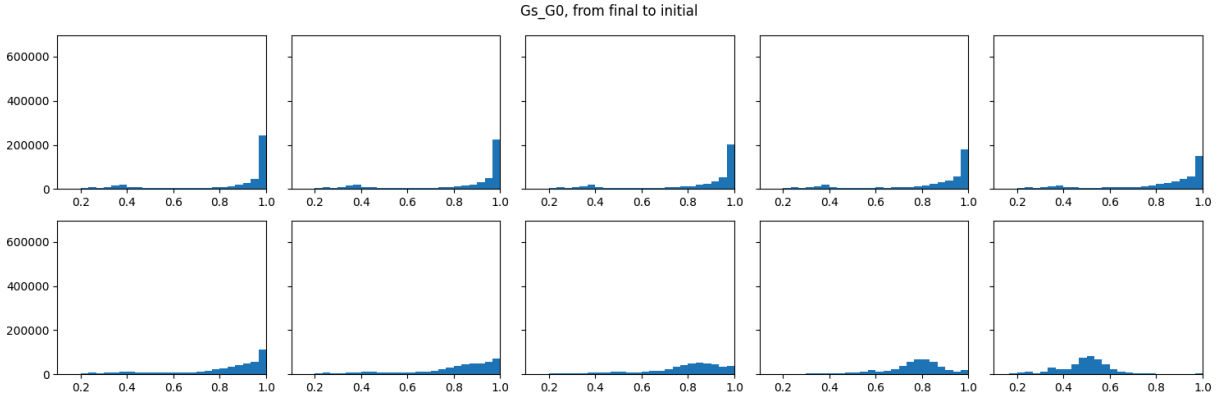
In [127…
```
Gs_Gt_infer_step_dist = process_dist(Gs_Gt_infer)
```

In [128…

```
def draw_step_dist(step_dist, name, x_range=[0.1, 1]):
    fig, axs = plt.subplots(2, 5, sharey=True, tight_layout=True, figsize=
(15, 5))
    fig.suptitle(f"{name}, from final to initial")
    nbins=30
    # We can set the number of bins with the *bins* keyword argument.
    for i in range(2):
        for j in range(5):
            axs[i][j].hist(step_dist[i*5 + j], bins=nbins)
            axs[i][j].set_xlim(x_range)
```

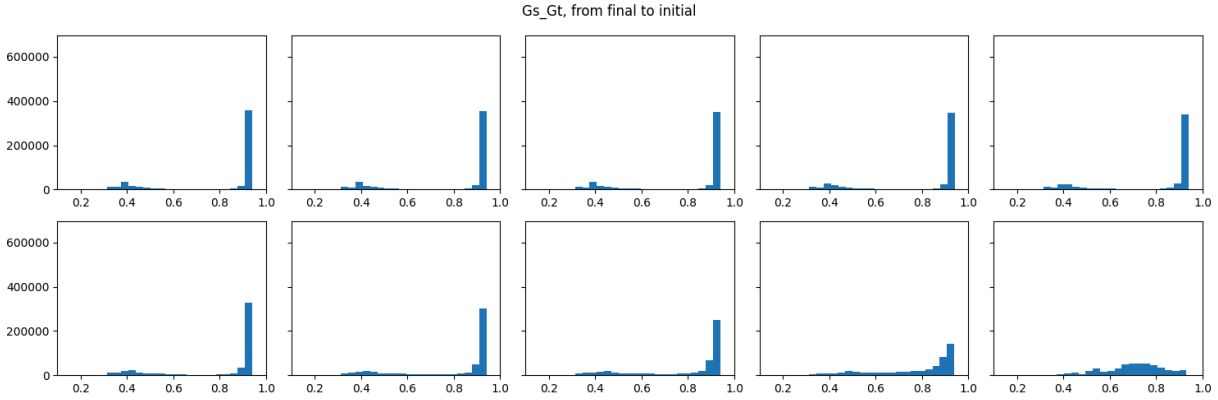
In [129…

```
draw_step_dist(Gs_G0_step_dist, 'Gs_G0')
```



In [32]:

```
draw_step_dist(Gs_Gt_step_dist, 'Gs_Gt')
```



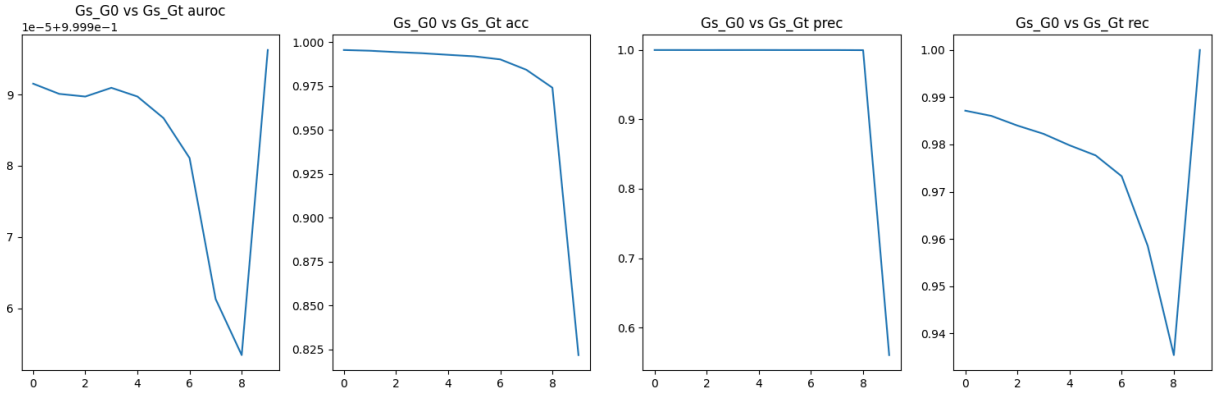
In [207…

```
def draw_metrics_step_by_step(y_true, y_logit, n_steps, true_name,
pred_name, true_thresh = 0.5, pred_thresh = 0.5):
    from sklearn.metrics import roc_auc_score, roc_curve
    auroc_list = []
    for i in range(n_steps):
        auroc = roc_auc_score(y_true[i] > true_thresh, y_logit[i])
        auroc_list.append(auroc)

    from sklearn.metrics import accuracy_score
    acc_list = []
    for i in range(n_steps):
        acc = accuracy_score(y_true[i] > true_thresh, y_logit[i] >
pred_thresh)
        acc_list.append(acc)
```

```
    from sklearn.metrics import precision_score
    prec_list = []
    for i in range(n_steps):
        prec = precision_score(y_true[i] > true_thresh, y_logit[i] >
pred_thresh)
        prec_list.append(prec)

    from sklearn.metrics import recall_score
    rec_list = []
    for i in range(n_steps):
        rec = recall_score(y_true[i] > true_thresh, y_logit[i] >
pred_thresh)
        rec_list.append(rec)

    fig, axs = plt.subplots(1, 4, tight_layout=True, figsize=(15, 5))
    axs[0].plot(auroc_list)
    axs[0].set_title(f"{true_name} vs {pred_name} auroc")
    axs[1].plot(acc_list)
    axs[1].set_title(f"{true_name} vs {pred_name} acc")
    axs[2].plot(prec_list)
    axs[2].set_title(f"{true_name} vs {pred_name} prec")
    axs[3].plot(rec_list)
    axs[3].set_title(f"{true_name} vs {pred_name} rec")
```

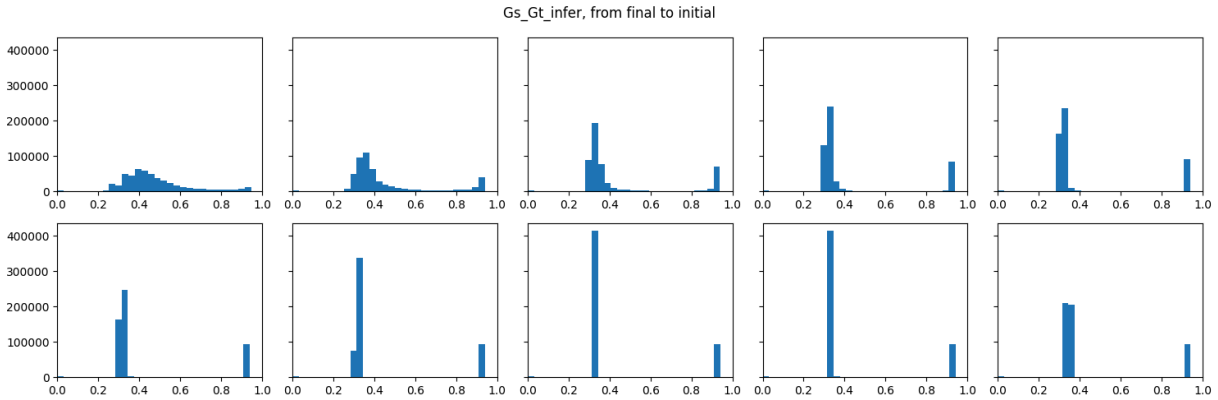
In [111…

```
draw_metrics_step_by_step(Gs_G0_step_dist, Gs_Gt_step_dist, 10, 'Gs_G0',
'Gs_Gt')
```



In [130…

```
draw_step_dist(Gs_Gt_infer_step_dist, 'Gs_Gt_infer', x_range=[0, 1])
```



In [378…

```
config = {
            'root': '/home/jiahang/DiGress/data/ND-code-
datasets/Application2-protein-contact-maps/full-predictions', 'norm':
'ND_norm', 'eps': 1e-2,
            'diffusion_steps': 20, 'force_reload': False,
            'slope': 50
        }
infer_data = ProteinInferDataset(**config)
gold_data = infer_data.golden
infer_idx, gold_idx = infer_data.data.idx, infer_data.golden.idx
gold_adj = to_dense_adj(gold_data.edge_index,
edge_attr=gold_data.edge_attr, max_num_nodes = gold_data.x.shape[0])[0]
NIMI_data = infer_data.data
NIMI_adj = to_dense_adj(NIMI_data.edge_index,
edge_attr=NIMI_data.edge_attr, max_num_nodes = NIMI_data.x.shape[0])[0]
NIMI_adj[torch.eye(NIMI_adj.shape[0]).bool()] = 0.

gold_adj_list = []
for _infer_idx in infer_idx:
    _gold_idx = gold_idx.index(_infer_idx)
    x_st, x_end = gold_data.slices['x'][_gold_idx], gold_data.slices['x']
[_gold_idx + 1]
    _cur_gold_adj = gold_adj[x_st:x_end, x_st:x_end].flatten().numpy()
    gold_adj_list.append(_cur_gold_adj)
gold_adj_dist = np.concatenate(gold_adj_list)
```

In [179…

```
NIMI_adj_list = []
x_st, x_end = NIMI_data.slices['x'][0], 0
for xslice in NIMI_data.slices['x'][1:]:
    x_end = xslice
    _cur_NIMI_adj = NIMI_adj[x_st:x_end, x_st:x_end].flatten().numpy()
    NIMI_adj_list.append(_cur_NIMI_adj)
    x_st = x_end
NIMI_adj_dist = np.concatenate(NIMI_adj_list)
```

In [122…

```
gold_adj_dist = np.expand_dims(gold_adj_dist,
0).repeat(Gs_Gt_infer_step_dist.shape[0], 0)
```

In [182…

```
NIMI_adj_dist = np.expand_dims(NIMI_adj_dist,
0).repeat(Gs_Gt_infer_step_dist.shape[0], 0)
```

In [192…

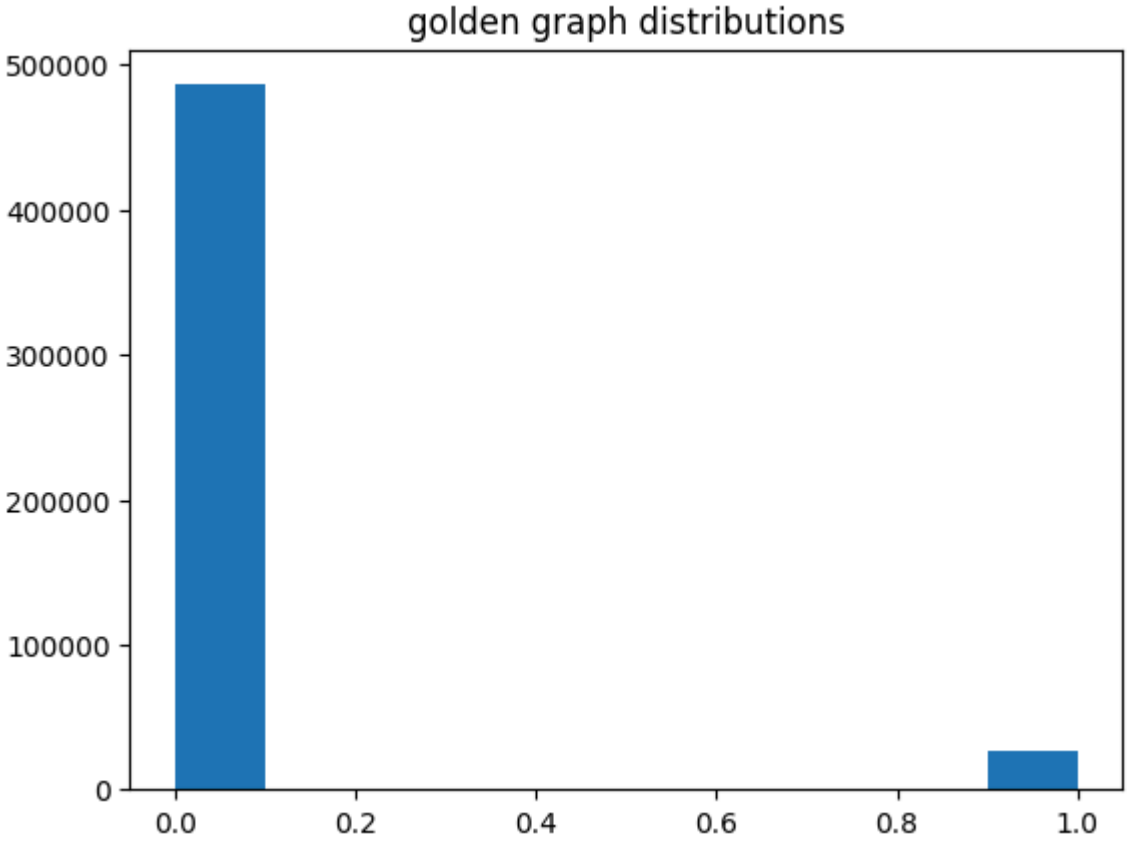
```
NIMI_adj_dist_slope_sigmoid = slope_sigmoid(torch.tensor(NIMI_adj_dist),
50).numpy()
NIMI_adj_dist_slope_sigmoid[NIMI_adj_dist == 0.] = 0.
```

In [133…

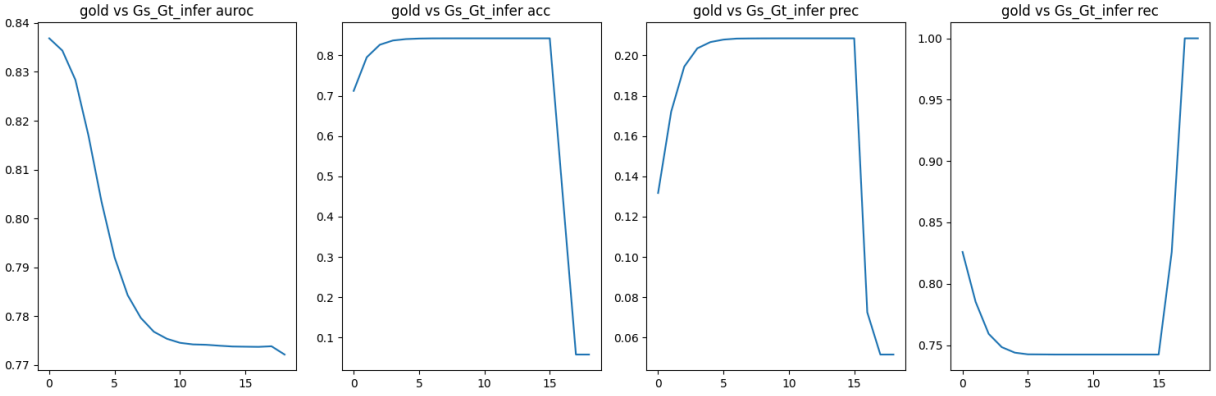
```
plt.hist(gold_adj_dist[0])
plt.title("golden graph distributions")
```

Out[133…

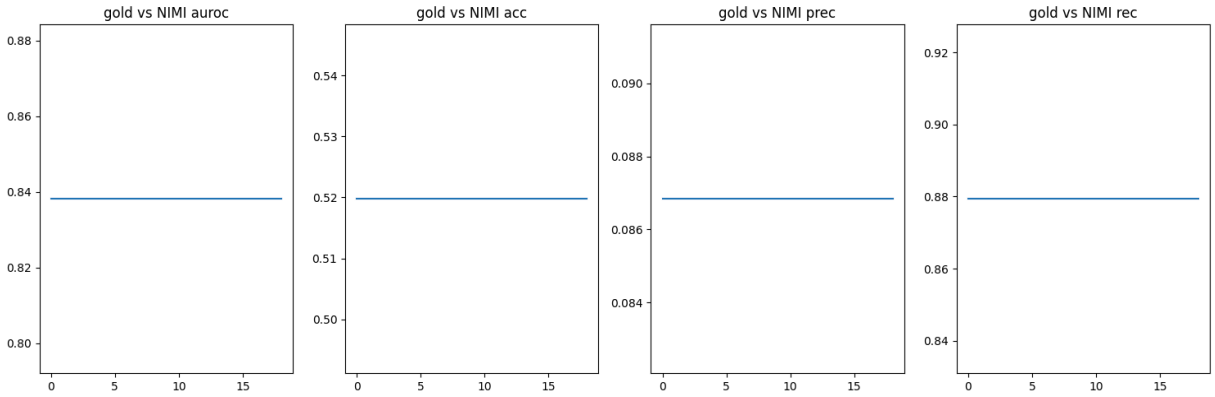
```
Text(0.5, 1.0, 'golden graph distributions')
```



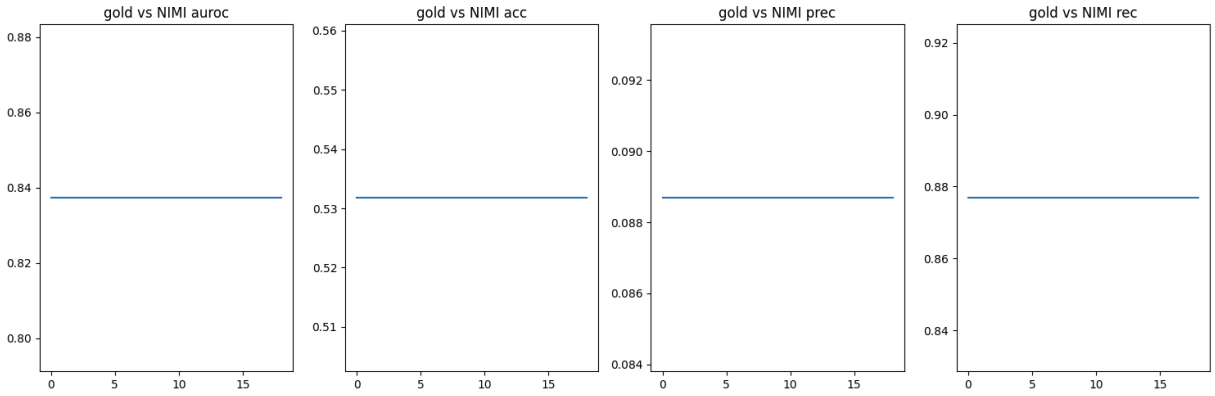
In [183…
```
draw_metrics_step_by_step(gold_adj_dist, Gs_Gt_infer_step_dist, 19,
'gold', 'Gs_Gt_infer')
```



In [195…
```
draw_metrics_step_by_step(gold_adj_dist, NIMI_adj_dist_slope_sigmoid, 19,
'gold', 'NIMI')
```



In [208…
```
draw_metrics_step_by_step(gold_adj_dist, NIMI_adj_dist, 19, 'gold',
'NIMI', true_thresh=0.5, pred_thresh=NIMI_adj_dist.mean())
```



Final evaluations

using original data without any preprocessing

NOTE symmetric adjacent matrix, should only get triu, more reasonable, better performance. NOW without triu, all elements.

In [210…
```
import os
```

In [309…
```
# original golden
root = '/home/jiahang/DiGress/data/ND-code-datasets/Application2-protein-
contact-maps/full-predictions'
files = os.listdir(root)
original_gold_E_list, original_gold_idx_list = [], []
original_gold_E_flatten_list = []
for filename in files:
    if not ('contact' in filename and '.mat' not in filename and '.pt' not
in filename):
        continue

    # get index
    idx = filename.split('contact_')[1].split('.npy')[0]
    original_gold_idx_list.append(idx)
    adj_path = os.path.join(root, filename)
```

```
    adj = np.load(adj_path)
    # adj = np.triu(adj)

    original_gold_E_list.append(adj)
    original_gold_E_flatten_list.append(adj.flatten())
```

In [310…

```
# original DI and MI
root = '/home/jiahang/DiGress/data/ND-code-datasets/Application2-protein-contact-maps/full-predictions'
files = os.listdir(root)
original_DI_E_list, original_DI_idx_list = [], []
original_DI_E_flatten_list = []
original_MI_E_list, original_MI_idx_list = [], []
original_MI_E_flatten_list = []
for filename in files:
    if 'ND' not in filename and ('DI' in filename or 'MI' in filename) and '.npy' in filename:
        idx = filename.split('I_')[1].split('.npy')[0]
        adj_path = os.path.join(root, filename)
        adj = np.load(adj_path)
        # adj = np.triu(adj)

        if 'DI_' in filename:
            # get index
            original_DI_idx_list.append(idx)
            original_DI_E_list.append(adj)
            original_DI_E_flatten_list.append(adj.flatten())
        elif 'MI_' in filename:
            # get index
            original_MI_idx_list.append(idx)
            original_MI_E_list.append(adj)
            original_MI_E_flatten_list.append(adj.flatten())
        else:
            print(filename)
```

In [292…

```
[_.min() for _ in original_DI_E_flatten_list]
```

Out[292…

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

In [342…

```
# model prediction file index list
root = '/home/jiahang/DiGress/data/ND-code-datasets/Application2-protein-contact-maps/full-predictions'
files = os.listdir(root)
pred_idx_list = []
for filename in files:
    if filename.endswith('.mat') or filename.endswith('.pt') or'_ND_' in filename \
        or ('DI_' not in filename and 'MI_' not in filename):
        continue
    idx = filename.split('.npy')[0]
    pred_idx_list.append(idx)
```

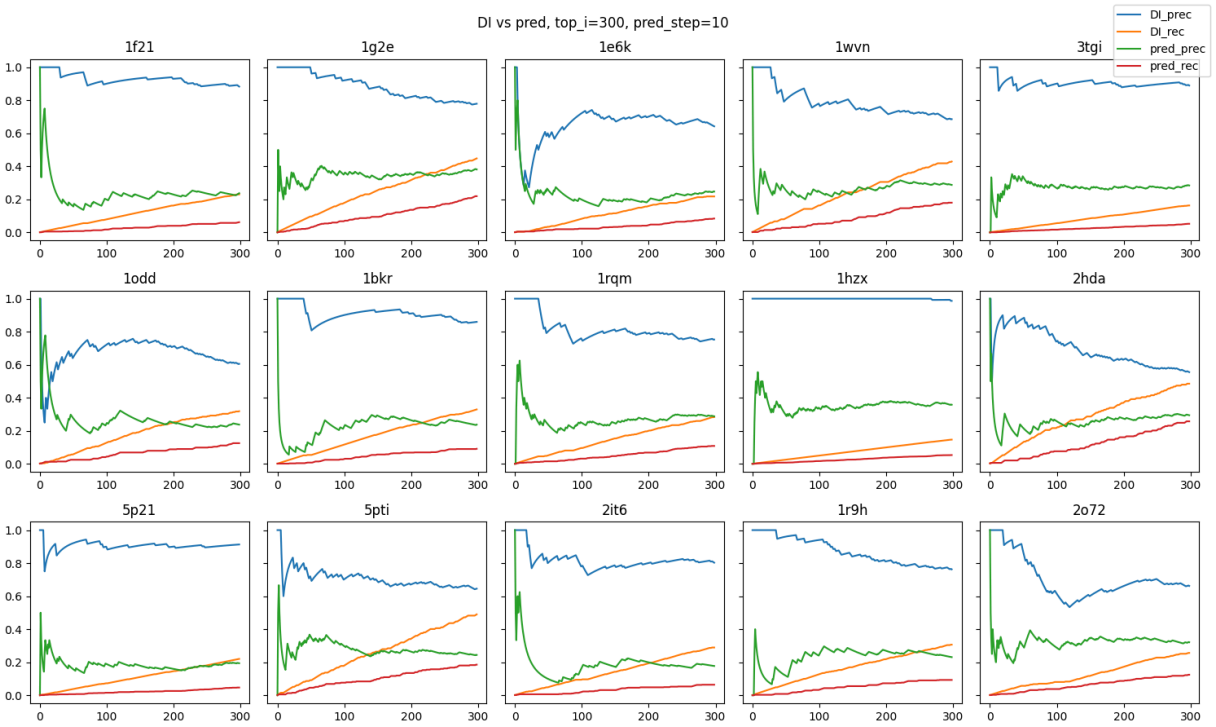
In [373…

```
def draw_prec_rec(option='DI', top_i=300, pred_step=-1):
    fig, axs = plt.subplots(3, 5, sharey=True, tight_layout=True, figsize=
```

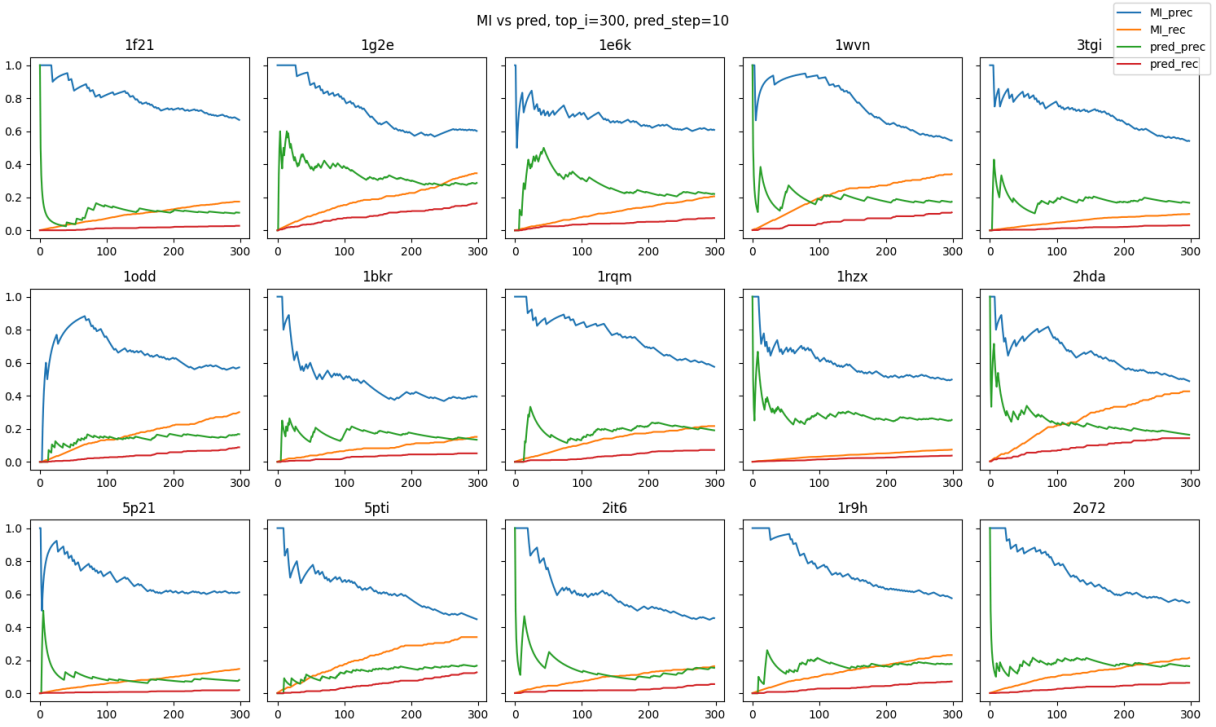
```
(15, 9))
    row, col = 0, 0
    for data_name in original_DI_idx_list:
        pred_data_name = option + '_' + data_name
        DI_data_idx = original_DI_idx_list.index(data_name)
        MI_data_idx = original_MI_idx_list.index(data_name)
        gold_idx = original_gold_idx_list.index(data_name)

        DI_data = original_DI_E_flatten_list[DI_data_idx]
        MI_data = original_MI_E_flatten_list[MI_data_idx]
        gold = original_gold_E_flatten_list[gold_idx]

        pred_data_idx = pred_idx_list.index(pred_data_name)
        e_st, e_end = NIMI_data.slices['edge_attr'][pred_data_idx],
NIMI_data.slices['edge_attr'][pred_data_idx + 1]
        pred_data = Gs_Gt_infer_step_dist[pred_step][e_st:e_end]

        DI_prec_list, DI_rec_list = [], []
        MI_prec_list, MI_rec_list = [], []
        pred_prec_list, pred_rec_list = [], []

        for _top_i in range(1, top_i):
            DI_exist_eidx = DI_data.argsort()[-_top_i:]
            DI_prec = gold[DI_exist_eidx].sum() / _top_i
            DI_rec = gold[DI_exist_eidx].sum() / gold.sum()

            MI_exist_eidx = MI_data.argsort()[-_top_i:]
            MI_prec = gold[MI_exist_eidx].sum() / _top_i
            MI_rec = gold[MI_exist_eidx].sum() / gold.sum()

            pred_exist_eidx = pred_data.argsort()[-_top_i:]
            pred_prec = gold[pred_exist_eidx].sum() / _top_i
            pred_rec = gold[pred_exist_eidx].sum() / gold.sum()

            DI_prec_list.append(DI_prec)
            DI_rec_list.append(DI_rec)
            MI_prec_list.append(MI_prec)
            MI_rec_list.append(MI_rec)

            pred_prec_list.append(pred_prec)
            pred_rec_list.append(pred_rec)
        # print(f"row: {row}, col: {col}")
        if option == 'DI':
            axs[row][col].plot(DI_prec_list, label='DI_prec')
            axs[row][col].plot(DI_rec_list, label='DI_rec')
        elif option == 'MI':
            axs[row][col].plot(MI_prec_list, label='MI_prec')
            axs[row][col].plot(MI_rec_list, label='MI_rec')
        axs[row][col].plot(pred_prec_list, label='pred_prec')
        axs[row][col].plot(pred_rec_list, label='pred_rec')

        axs[row][col].set_title(data_name)
        if row == 2 and col == 4:
            handles, labels = axs[row][col].get_legend_handles_labels()
            fig.legend(handles, labels, loc='upper right')
```

```
        if col == 4:
            row += 1
            col = 0
        else:
            col += 1
    plt.suptitle(f"{option} vs pred, top_i={top_i}, pred_step=
{pred_step}")
```

In [374…
```
draw_prec_rec(pred_step=10)
```

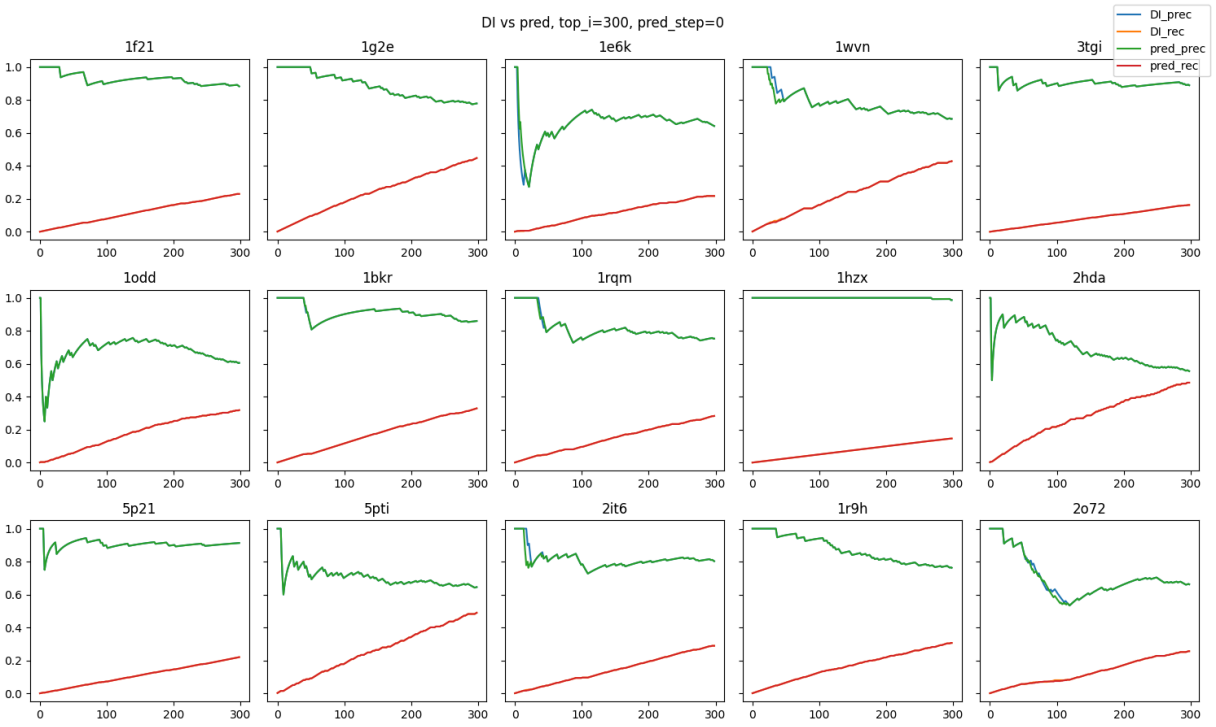


In [375…
```
draw_prec_rec(option='MI', pred_step=10)
```



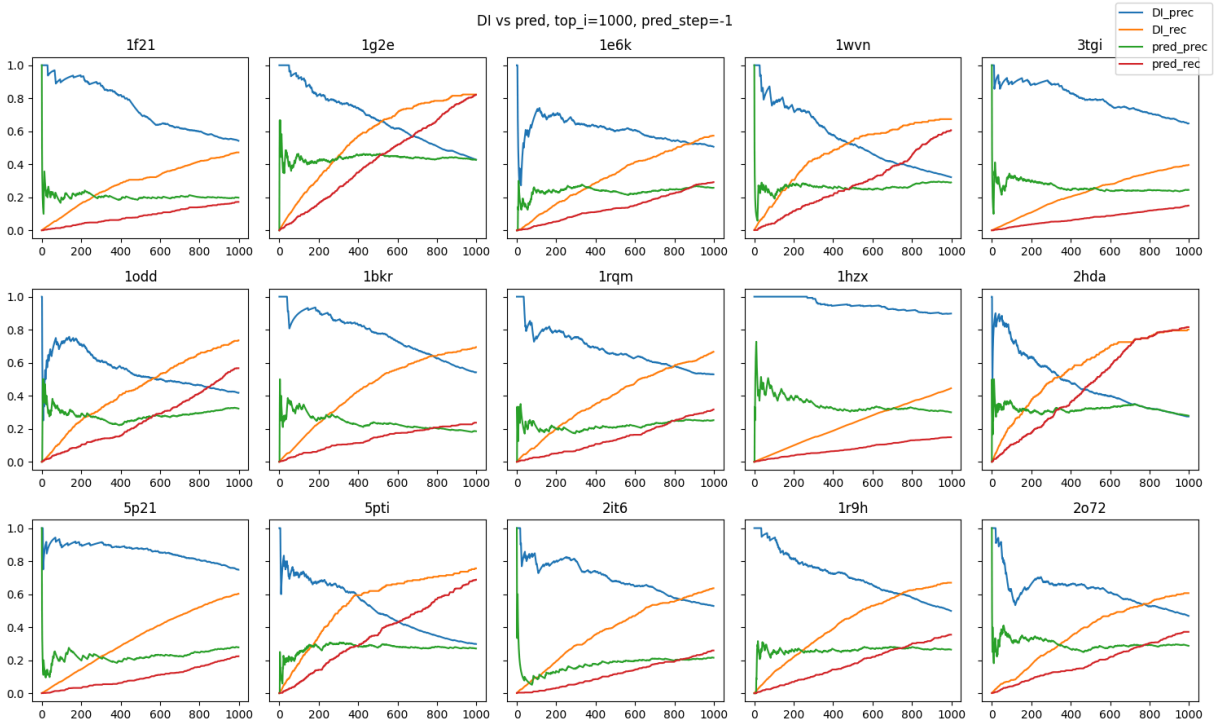
In [376…

```
draw_prec_rec(option='DI', top_i=300, pred_step=0)
```



In [377…

```
draw_prec_rec(option='DI', top_i=1000, pred_step=-1)
```


DI vs pred, top_i=1000, pred_step=-1
DI_prec
DI_rec
pred_prec
pred_rec
1f21
1g2e
1e6k
1wvn
3tgi
1odd
1bkr
1rqm
1hzx
2hda
5p21
5pti
2it6
1r9h
2o72

In [ ]: