METIS

南京邮电大学 Nanjing University of Posts and Telecommunications

一．METIS默认值说明

名称	选项说明	默认值	默认值说明
METIS_OPTION_PTYPE	图划分方法类型	METIS_PTYPE_KWAY	多层次k-way划分
METIS_OPTION_OBJTYPE	划分目标函数类型	METIS_OBJTYPE_CUT	边割数量最小化
METIS_OPTION_CTYPE	粗化阶段匹配策略	METIS_CTYPE_SHEM	按边权排序的重边匹配
METIS_OPTION_IPTYPE	初始划分策略	METIS_IPTYPE_GROW	贪心法扩展划分
METIS_OPTION_RTYPE	细化阶段算法	METIS_RTYPE_FM	基于FM算法

二．FM算法

FM算法是对K-L算法的改进，用于加速图的划分过程，尤其适合于大规模稀疏图的局部优化。

METIS

1. 算法思想

- FM算法试图在**两个子图之间交换节点**，以**最小化边割（cut size）**，同时**保持子图大小的平衡。**
- 它以**线性时间复杂度**（O(n)）进行局部优化，主要流程为：
- 初始化两个划分区域A和B。
- 计算每个节点的“增益”值：将该节点从当前划分移动到另一侧能减少多少边割。
- 使用优先级队列来管理增益最大的节点。
- 在不违反平衡条件的前提下，依次选择增益最大的节点进行移动，并锁定。
- 记录增益变化过程中的最大累计增益位置。
- 恢复到该最佳状态，完成一轮优化。
- 多轮迭代直到无法再优化。

2. 增益公式

如果节点v有外部边（连到另一划分）为E_{ext}，内部边为E_{int}，则增益： $gain(v) = E_{ext}(v) - E_{int}(v)$

BAGE

一．潜变量z作为asn embedding效果实验

数据集：20170426，20220328异常事件

dataset	Excepted origin-AS	Detected origin-AS	bgpvector cosine similarity	z cosine similarity
20170426	41268	12389	0.42	0.28
20220328	13414	8342	0.49	-0.09

实验结果分析

使用BAGE生成的潜变量z作为asn embedding，计算监测起源AS与意外起源AS的余弦相似度，比bgpvector所用的asn embedding，结果更小，说明更容易检测到异常。

BAGE

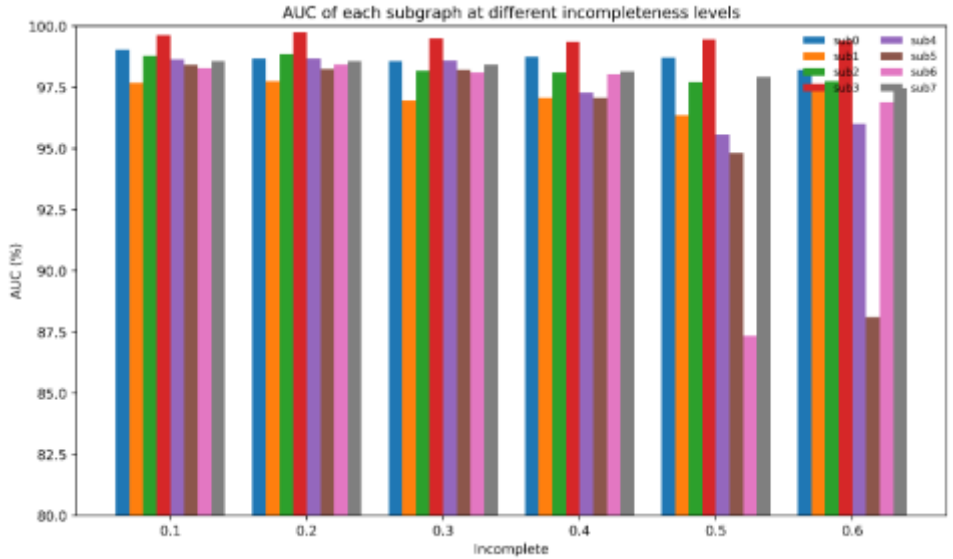
南京郵電大學 Nanjing University of Posts and Telecommunications

二. BAGE不完整度实验

数据集：bview.20250401.0000.txt

	0.1	0.2	0.3	0.4	0.5	0.6
子图0	99.04	98.69	98.58	98.75	98.71	98.22
子图1	97.69	97.75	96.97	97.07	96.37	97.59
子图2	98.80	98.85	98.18	98.11	97.71	97.75
子图3	99.63	99.75	99.50	99.35	99.45	99.41
子图4	98.64	98.69	98.59	97.28	95.57	96.01
子图5	98.41	98.25	98.21	97.07	94.81	88.12
子图6	98.30	98.44	98.11	98.05	87.35	96.88
子图7	98.56	98.57	98.43	98.16	97.92	97.48

BAGE



实验结果分析

不完整度的增加有auc减少的趋势，但auc的实验结果整体较高。

BAGE

三. BAGE内存使用实验

节点个数	最大内存使用
10000	7665.42 MiB
20000	17664.99 MiB
30000	跑不动

实验结果分析

该实验证明BAGE节点数量对于内存消耗的影响，以证明图分割的必要性。

ADA-GAD

南京郵電大學 Nanjing University of Posts and Telecommunications

一. ADA-GAD实验

实验运行情况如下图所示

```
2025-05-01 20:44:31,535 - INFO - Using best configs
------ Use best configs ------
Namespace(T_f=2, activation='relu', aggr_f='min', all_encoder_layers=0, alpha_f=0.9, alpha_l=3, attention=2, attn_drop=0.1, attr_decoder='gcn', attr_decoder_num_layers=1, attr_encoder='gcn
', attr_pretrain_uncertainty_rate=0, attr_remask_uncertainty_rate=0, batch_size=32, concat_hidden=False, dataset='books', deg4feat=False, device=0, drop_edge_rate1=0, drop_edge_rate2=0.01,
 drop_edge_rate3=0, drop_path_length1=3, drop_path_length2=3, drop_path_length3=3, drop_path_rate1=0, drop_path_rate2=0.01, drop_path_rate3=0.01, dropout_f=0, each_pu_epoch=15, edge_encode
r_num_layers=2, eta=1.0, in_drop=0.2, load_model=False, logging=False, loss_f='add_log_t_entropy', loss_fn='mse', loss_weight_f=-1, lr=0.001, lr_f=0.01, mask_rate1=0.01, mask_rate2=0, mask
_rate3=0, max_epoch=20, max_epoch_f=20, max_pu_epoch=45, model_name='ADANET', negative_slope=0.2, node_encoder_num_layers=2, norm=None, num_heads=1, num_hidden=12, num_layers=2, num_out_he
ads=1, optimizer='adam', pooling='mean', predict_all_edge1=0, predict_all_edge2=0.1, predict_all_edge3=0.1, predict_all_node1=False, predict_all_node2=False, predict_all_node3=False, repla
ce_rate=0.3, residual=False, sano_weight=1, save_model=False, scheduler=True, seeds=[0], select_gano_num=30, sparse_attention_weight=0, struct_decoder='gcn', struct_decoder_num_layers=1, s
truct_encoder='gcn', struct_pretrain_uncertainty_rate=0, struct_remask_uncertainty_rate=0, subgraph_encoder_num_layers=2, theta=1.0, topology_encoder='gcn', use_cfg=True, use_encoder_num=3
, use_nni=False, use_ssl=1, walks_per_node1=1, walks_per_node2=1, walks_per_node3=3, warmup_steps=-1, weight_decay=0.0002)
####### Run 0 for seed 0
======== train attr encoder ========
2025-05-01 20:44:31,627 - INFO - Use schedular
2025-05-01 20:44:31,627 - INFO - start training..
D:\Projects\Python\ADA-GAD\E_high.py:5: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_
(True), rather than torch.tensor(sourceTensor).
  adj_tensor = torch.tensor(adj_matrix, dtype=torch.float32)
D:\Projects\Python\ADA-GAD\E_high.py:21: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad
_(True), rather than torch.tensor(sourceTensor).
  deg_matrix = torch.diag(torch.sum(torch.tensor(adj_matrix, dtype=torch.float32), dim=1))
======== train struct encoder ========
2025-05-01 20:44:36,182 - INFO - Use schedular
2025-05-01 20:44:36,182 - INFO - start training..
======== train topology encoder ========
2025-05-01 20:44:42,429 - INFO - Use schedular
2025-05-01 20:44:42,429 - INFO - start training..
finish one train!
auc_score: 0.6462
# final_auc: 64.62±0.00
```

ADA-GAD

二. ADA-GAD论文

1. 问题

传统的基于重建的图神经网络（GNN）方法在图异常检测中取得了显著成功，但它们在面临图中的异常模式时会遇到“异常过拟合”和“同质性陷阱”问题。

异常过拟合：GNN模型在稀疏图中容易过拟合异常节点的特征，导致异常节点被错误重建。

同质陷阱：GNN的同质性假设使得异常节点的邻居正常节点难以被准确重建，放大误差。

ADA-GAD

2. 方法

ADA-GAD框架包含两个主要阶段：

- **第一阶段：异常去噪预训练**

1. 量化图的异常程度：

定义基于图信号高频能量的指标：

属性异常幅度： $A_{ano}(\mathcal{G}) = E_{high}(\mathcal{G}, X) = \frac{X^T L X}{X^T X}$

结构异常幅度： $S_{ano}(\mathcal{G}) = E_{high}(\mathcal{G}, D) = \frac{D^T L D}{D^T D}$

图的异常幅度： $Gano(G) = Aano(G) + Sano(G)$

为了生成去噪图，在增强预算范围内最小化图的异常幅度。

ADA-GAD

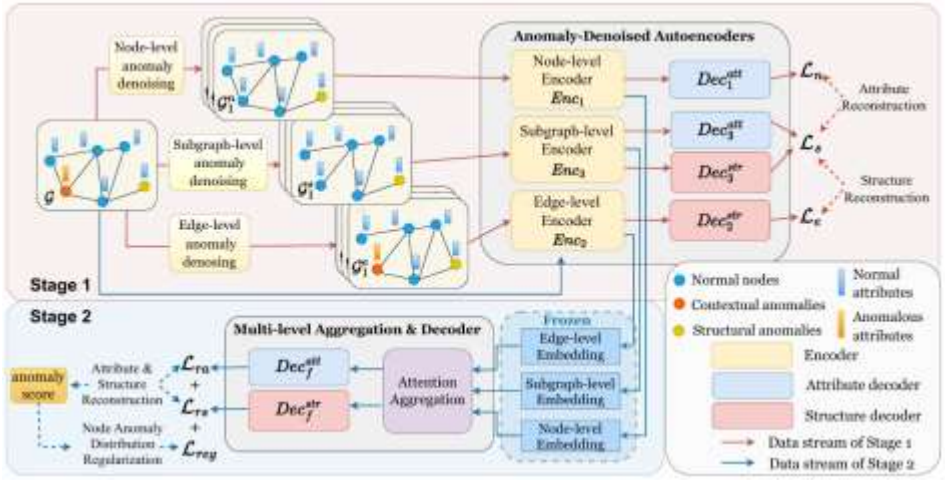
2. 多级去噪预训练：

采用三种掩码策略来生成增强图并执行去噪预训练

节点级：随机替换或置零节点特征，重构特征。

边级：随机掩码边，即从邻接矩阵中删除，重构结构。

子图级：基于随机游走策略采样子图，，并在其上同时执行节点和边的掩码，重构属性和结构。



Overall framework of ADA-GAD

ADA-GAD

- **阶段二：重训练用于检测**

在此阶段中，不再对图进行掩码操作，而是固定第一阶段预训练得到的图编码器，舍弃预训练的解码器，并从头训练两个统一的解码器（一个用于属性，一个用于结构），以对原始图中的异常信息进行检测。

1. 多层次嵌入聚合

预训练阶段中，得到了三个层次（节点级、边级、子图级）的嵌入表示。首先通过全连接层对其进行处理，然后利用注意力机制对其进行聚合，得到每个节点的最终多层次嵌入表示h。

2. 图重构与异常检测

聚合后的多层次嵌入h输入到属性解码器Dec_{att}^{f}和结构解码器Dec_{str}^{f}中，对原始图的邻接矩阵A和属性矩阵X进行重构，分别记作$\hat{A}$和$\hat{X}$。

对应的重构损失函数为：$L_{rec}=(1-\gamma)\cdot\left\|A-\hat{A}\right\|_F^2+\gamma\cdot\left\|X-\hat{X}\right\|_F^2$

每个节点s_i的异常分数定义为其结构和属性重构误差的加权和：$s_i=(1-\gamma)\cdot\|a_i-\hat{a}_i\|^2+\gamma\cdot\|x_i-\hat{x}_i\|^2$

其中a_i，$\hat{a}_i$分别是节点的原始和重构结构向量，x_i，$\hat{x}_i$是其原始和重构属性向量。

ADA-GAD

3. 节点异常分布正则项

利用节点的异常分布损失对模型进行正则化，并通过提升异常分布的熵值来增加重构的难度。设节点v_i 的熵为S_i：

整个图的正则化项定义为：$L_{reg} = -\sum_{v_i \in V} S_i$

4. 优化目标

最终优化的总损失函数为：$L = L_{rec} + \gamma_{reg} \cdot L_{reg}$

基于这种损失，重新训练了聚合和解码器模块。训练完成后，根据每个节点的异常分数进行排序，并根据给定的异常率选取异常节点。