这是一篇AAAI2022上的一篇关于无监督图表示学习的文章，与其他的图表示学习方法相比，这篇文章的一个亮点就如他的题目一样，他用了一种相对来说更简单的方法，而这种简单的方法在节点分类的准确性上也做到了sota的结果，并且非常的节约时间。

文章首先介绍了现有的无监督图表示学习方法。现存的无监督图表示学习方法大多是基于GNN的，而这些方法中比较有代表性的一种就是对比学习，而对比学习的一个主要任务就是定义我们的输入内容和与它进行对比的内容。比如说，DGI模型将节点表示和图的概括表示进行对比并最大化它们的相关信息，GMI用输入图和输出图的表示进行对比，GRACE和GCA它们从多个视角对同一张图中的每个节点进行表示，并希望能够最大化每个结点在不同视角下的相关性。

这些方法已经在许多任务上被证明是有效的，但是作者认为这些方法的很大精力都花在了如何对原始数据进行数据增强，而数据增强会在训练过程中会花费大量的时间，因此效率不是很高；除此之外，现存的这些方法对于特征维度是敏感的，模型的效果往往会随着特征维度的增长而增长，因此模型难免会使用比较高维的特征，而这也带来了更大的计算开销；另外，很多的对比学习方法在进行对比的时候往往会设计一个discriminator来衡量两者之间的相似性，这个操作在训练过程中也花了比较多的时间。因此，作者这篇工作的重心就是采取有效的方法来简化模型，从而减少训练过程的时间花销并且保持模型的能力。

然后我们具体来看下他的做法。首先他使用了最简单的方法来表示一张图，即由每个节点的特征向量组成一个特征矩阵X，以及表示节点之间联通情况的邻接矩阵。然后是对比学习任务的设计。作者首先设计了一个anchor embedding，也就是一张图它的基准表示，代表了这张图的语义信息，此前的方法在设计anchor embedding的时候都会用到GCN的方法，比如说有的方法它会使用先使用GCN学到对于每个节点的表示，然后通过一个函数整合节点的信息得到整张图的表示，作为anchor embedding。但作者认为GCN相对来说是比较花时间的，所以为了节约时间，作者直接使用了一个多层感知机来得到一张图的anchor embedding，可以看到他这里将初始特征矩阵输入多层感知机中，最后得到一个anchor embedding H。

Anchor embedding产生之后，接下来就是与他进行对比的内容的设计。作者在这里同时设计了negative embedding和postive embedding，也就是负嵌入和正嵌入，然后之后同时将它们与anchor embedding 进行对比，我们希望negative embedding与anchor embedding之间的距离尽可能远，而postive embedding 和 anchor embedding 之间的距离尽可能近。Negative embedding的思想其实是比较常见的，作者在这里主要是对negative embedding的方法进行了简化，在之前的工作中，negative embedding往往可能需要对初始图进行一定程度的破坏，然后在被破坏的图上进行GCN操作得到negative embedding，作者认为这种方法相对来说是比较耗时的，所以在这里作者对negative embedding的方法进行了简化，他直接把我们得到的anchor embedding在行的维度上进行打乱，就得到了negative embedding。

负嵌入结束之后，我们还需要进行正嵌入。现存的方法往往使用数据增强方法来获得丰富的信息从而进行对比学习，但数据增强方法相对来说比较耗时。因为数据增强的目标是获得更为丰富的信息，所以为了达到这个目标作者采用了更为简单的方法，也就是这里提到的postive embedding，而postive embedding在这篇文章中又分为了两种，即structural embedding 和 neighbor embedding 他们分别包含了原始图的结构信息和邻居信息。在获取structural embedding的时候，作者直接使用了原始的GCN，而且他这里使用的参数W和之前进行anchor embedding的W是共享的，这也进一步减少了计算的开销。除了获取structual embedding作为postive embeding，他这里还获取了neighbor embedding的信息作为第二种postive embedding，对于邻居节点的信息，他这里采用的方法也很简单，对于每个节点来说，他都只采样m个邻居节点，并且直接加和得到neighbor embedding。之所以既要获得structural embedding和neighbor embedding，作者他是这样解释的，他认为structual embedding得到的是全局的一般的信息，而neighbor embedding得到的是局部的特别的信息，他们从不同的角度提供信息，因此可以做到互补。

以上，对比学习中embedding的设计就完成了，接下来是损失函数的设计。他这里的思想和transe中的负采样的思想是有点像的，我们希望negative embedding与anchor embedding之间的距离比postive embedding与anchor embedding之间的距离更大，这里的d可以直接使用l2范数，阿尔法是我们调节的超参数。而由于前面提到，这篇文章中作者设计了两个postive embedding，一个表示结构信息，一个表示邻居信息，所以这里的损失函数也需要加他们各自的损失加起来。但作者认为，做到这一步是还不够的，因为在这里我们只保证了negative的距离比postive的距离更大，但是并没有做到让postive embedding与anchor embedding之间的距离尽可能地小，如果postive embedding与anchor embedding之间的距离过大的话，这样做对比学习是不合适的。所以作者希望能够对postive embedding与anchor embedding之间的距离进行约束，从公式6我们可以看出，postive的距离的上限是由negative的距离决定的，所以，如果我们对negative的距离进行约束的话，那么postive的距离也会被约束，因为作者的整体的想法就是，我虽然希望negative的距离比较大，但是我又不希望太大，从而保证我在有一定区分度的情况下还能使得postive embeding的信息和anchor embeding之间的信息是尽可能匹配的，因此，它对negative的距离增加了一个上限，这个β是我们我可以调节的超参数。而根据这个不等式，作者在损失函数中又新增了一项LU，它就约束了训练过程中negative距离的上限，从而保证postive的信息与anchor的信息是接近的。最终，它的损失函数就长成了这个样子，其中对于LS和LN还设置了权重超参。

之后就是实验部分，作者在8个数据集上进行了节点分类任务，这8个数据集包括两个3个引文网络数据集，2个亚马逊销售数据集以及3个大规模数据集。在baseline上使用了一个传统算法deepwalk，两个半监督算法GCN和GAT，以及8个无监督算法。结果如表1和表2所示，可以看出作者提出的SUGRL达到了SOTA的效果，并且它的时间消耗是非常的低的，训练速度基本排在第一的水平，与其他的无监督方法相比，比之前最快的GIC模型平均快了4.4倍。

随后作者进行了消融实验，首先作者讨论了结构信息、邻居信息以及upper bound设置对于模型区分能力的影响。他这里使用的指标是，同类信息的差异度与异类信息的差异度比值，具体的做法作者在这里没有细讲，他给出了最终的结果。对于完整的实验设置来说，随着训练epoch的增加，差异度的比值逐渐下降并最终保持在一个很低的水平，说明同类信息的差异度下降以及异类信息的差异度上升，从而导致它们之间的比值很小。而如果我们去掉训练中用到的结构信息或者是邻居信息或者是取消negative 距离的上限设置，这个差异度的比值最终都是比较大的，说明同类信息的差异度比较大或者是异类信息的差异度比较小。从而证明了这三个因素都是有效的。

之后，作者也对loss函数的三个部分也进行消融实验，具体结果如表3所示，去掉其中任何一个loss子函数，模型效果都会出现下降。

消融实验结束之后，作者又对超参数的取值进行了讨论，首先对于loss函数里用到的阿尔法和贝塔，作者在0.1到0.9的取值范围内对他们进行讨论，发现当阿尔法和贝塔的取值都比较大的时候，模型效果是比较好的，对此作者认为如果阿尔法和贝塔太小，那么正类和负类之间的差异不够，模型的区分能力就不足。随后，作者也对LS和LN的权重也进行了讨论，发现如果LS和LN太小，那么模型的能力也会下降，说明这两个子函数都是重要的，因为他们能够有效的把postive embedding和negative embedding区分开。

最后，作者对于这篇工作的高效性进行了总结。首先，在他们的工作里，他们没有使用到传统的数据增强方法，而是使用更为快速的GCN方法提取结构信息并直接聚合邻居信息，时间消耗很少；第二点是，他们的嵌入速度很快，比如在获取anchor embedding的时候它们使用了比GCN更快的多层感知机，而且在获取邻居信息的时候，它们也没有聚合全部的邻居信息而是只采样一部分邻居，这也被证实是快速而有效的。第三点，他们的工作能够在低维的特征维度上就获得很好的性能，一般的工作可能需要512维才能达到比较好的效果，但是在这个实验里，作者发现模型效果对于特征维度是不敏感的，在128维时就可以达到很好的效果了。第四点是，他们没有设计复杂的discriminator，而discriminator在其他的工作里是比较常用且比较耗时的，作者只通过设计损失函数就达到了比较好的区分效果，且花费的时间更少。

以上就是这篇文章的全部内容。