深度学习模型计算图相同子结构的识别和展示

“ 本项目受到 “开源软件供应链点亮计划-暑期2020” 的资助和支持,由peter@mail.loopy.tech设计开发，由gaohan19@huawei.com指导。 ”

1. 简介

MindSpore是华为自研的深度学习框架，其中的计算图模式是一种业界主流的用来进行数据传递与计算的形式。计算图主要包含节点和有向边，节点表示计算和控制操作，边表示数据的流向和控制等关系。计算图的高效合理展示，有助于用户更好的理解模型结构、发现和调试模型训练过程中出现的问题。然而，大型深度学习模型往往有着复杂的计算图结构，包含有成千上万个节点和更多的边。在这些点和边之中，包含有许多结构相同或高度相似的子结构，这些子结构不仅从图的拓扑结构上，甚至从深度学习语义上具有高度的相似性。快速识别大型计算图中上述的相同子结构，能够支持后续用收折、重叠等方式大幅减少页面中同时呈现的节点和边的数目，从而大幅改善计算图的展示效果。

2. 为什么需要新算法？

当前应用情景与常规子图挖掘大不相同：

- 度量方式：支持度变得不那么重要，而应以压缩输入图的程度来度量
- 单图挖掘：在单个大图中寻找子图，而不是多个小图
- 有向无环图：计算图是有向无环图
- 无边权：计算图的所有边都是相同的
- 算法级并行：在并行场景下能快速稳定执行
- 层次化算子节点：计算图中的各个算子节点是具有层级的

由此，我们基于Apriori思想改进了现有的频繁子图挖掘算法，并使用Python进行了编码实现

3. 算法简述

本项目分为两个模块，分别用于形成节点集和在特定节点集中进行子图挖掘。

整个算法基于一个基本的事实规律：**频繁子图的任意子图或子节点都是频繁的**。由此，我们只需要找到频繁的节点，并由频繁的节点开始进行聚合和生长，就能找到需要的频繁子图。

3.1. 节点集形成算法

节点集形成算法就是从所有节点中筛选出某个特定层级的节点，形成待挖掘的节点集。主要难度在于，各个节点存储的上下游节点都是level-1的节点，为了在筛选中保持图的结构，需要将上下游节点重定向至特定层级的节点，需要递归的查询各个节点的符合要求的祖先。

右侧给出了算法伪代码：

Algorithm 1: 形成第 i 层的节点集

Input: 图 G

Output: 某层的节点集

为图中所有种类 (包括 Scope) 的节点分配独立 ID;

按种类进行分类，获得 Scope 节点集 S_{scope}，算子节点集 S_{normal}，参数和常数节点集 $S_{parameter}$;

确定所有节点的层级;

if $i==1$ **then**

  输出 $S_{normal}+S_{parameter}$;

else

  筛选 S_{scope}，获得层级 i 的节点集 $S_{scope-i}$;

  for $S_{scope-i}$ 中的节点 n **do**

    重定向 n 的上下游节点 id 至层级 i;

  end

  输出 $S_{scope-i}$;

end

3.2. 子图挖掘算法

本项目所使用的子图挖掘算法，基于Apriori思想进行改进，使用“自下而上”的方法，首先生成单节点的核，核进行多次生长，每次生长后移除原来的核，当所有核都生长为子图集后，算法终止。

右侧给出了算法伪代码：

Algorithm 2: 子图挖掘

Input: 节点集 S_{node}
Output: 子图集的集合 $S_{subgraph}$
统计节点集 S_{node} 中的频繁节点;
将频繁节点标记为单节点的核，放入计算池中;
while 计算池不为空 **do**
  从计算池中取出一个核 C_i;
  for Ci 的子图模式中的边界点 P_{ij} **do**
    for Ci 的所有子图实例中 Pij 位置上的对应节点 N_{ijk} **do**
      记录节点 N_{ijk} 的所有下游节点的种类;
    end
    统计节点 N_{ijk} 的下游节点的种类频次;
    for 节点 N_{ijk} 的下游节点的种类 T_{ijkl} **do**
      if 生长后的新核 C_{ijkl} 满足各种阈值条件，且未被注册 **then**
        注册该新核 C_{ijkl};
        将新核放入计算池;
      end
    end
  end
  if 无法生长出新核，且 C_i 是有效子图集 **then**
    将核 C_i 的拷贝放入子图集的集合 $S_{subgraph}$ 中;
  end
  删除 C_i;
  for $S_{subgraph}$ 中的子图 SG_i **do**
    for $S_{subgraph}$ 中除 SG_i 的其他子图 SG_j **do**
      if SG_i 是 SG_j 的子图，且 SG_i 的各项参数不符合带罚项的阈值条件 **then**
        移除 SG_i;
      end
    end
  end
end
输出 SG_i;

4. 实现

以下简述算法的实现的相关思路和方式

4.1. 数据结构

4.1.1 节点

节点由`SNode`类所定义的数据结构存储，每个节点既保存了上下游节点（保存计算图中的联系），也保存了自己所属的命名空间（保存节点树中的联系）。`Scope`类继承自`SNode`，在其基础上，`Scope`还额外存储了自己的组成节点。为了避免循环引用，`SNode`和`Scope`在存储其它节点时，只存储了ID，具体节点信息需要到`SMSGraph`中去查询。同时，它们的设计采用了鸭子类型的风格，在子图挖掘阶段，两种类型的实例是等效的。

4.1.2 计算图

计算图由 `SMSGraph` 类所定义的数据结构存储，它保存了一个计算图中的全部信息，并提供了丰富的接口以接受各种查询。它的另一个主要功能是解析 `MSGraph` 对象，将其数据按本项目设计的方式整理和清洗。

子图的层次结构没有专门的类定义存储，而是使用节点存储的信息结合计算图的查询接口完成查询。在需要递归建树或查询时，使用猴子补丁的方式，将函数临时附加到节点类上，以完成需要的功能。

4.1.4 子图

子图相关信息由`Subgraph`类所定义的数据结构存储（生长时为`SubgraphCore`），与大多数编码实现不同，本项目中的图没有保存边的关系，而是按序保存了节点的关系，即子图模式。在子图挖掘中，核（子图核集）为核心数据结构，所有调度和控制都围绕着核展开。比如`Node1(biaAdd)->Node2(Conv2D)`与`Node3(biaAdd)->Node4(Conv2D)`同构，则子图核集保存的是`{pattern:['biaAdd', 'Conv2D'], nodes:[(1,2), (3,4)]}`，ID为`hash("1-2")`.为了保证核模式在遍历时只遍历边界点，并减少功能耦合，核可以作为迭代器，只有边界点会被遍历取得。

4.2. 并行调度与执行

为了充分使用硬件并加快速度，子图挖掘部分算法执行采用Map-Reduce机制，执行器中的计算池存储着前一生长周期被提出的核，通过线程池调度的方式进行生长计算，从而获得下一生长周期的新计算池。每个核的生命周期都为一个生长周期，在生长周期结束后可能被销毁或拷贝提交。同时使用了子图核集的注册机制以避免冗余计算，每一子图核集(除单节点的外)在被提出时都需要通过执行器进行注册(注册需要是线程安全的)，冗余的核将不被加入计算池。

4.3. 用户使用

```
(env) ~ » detect-subgraph -h
usage: detect-subgraph [-h] [-v] [--verbose] [--safe-mode] [-w MAX_WORKER]
                       [-i MIN_SUBGRAPH_INSTANCE_NUMBER]
                       [--min-nodes MIN_SUBGRAPH_NODE_NUMBER]
                       [--max-nodes MAX_SUBGRAPH_NODE_NUMBER]
                       [-p SUB_SUB_GRAPH_THRESHOLD_PENALTY]
                       [--skipped-level SKIPPED_LEVEL] [-c]
                       [--disable_scope_boundary] [-d]
                       graph path result path

Detect subgraphs in a Mindspore computational graph

positional arguments:
  graph path            The path of the pb file where the whole graph are
                        stored
  result path           The path of json file where the detected subgraphs
                        should be dumped.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  --verbose             Print details to console
  --safe-mode, -s       Do some extra computation to make sure safety
  -w MAX_WORKER, --worker MAX_WORKER
                        The worker number of Thread Pool, -1 = cpu_count
  -i MIN_SUBGRAPH_INSTANCE_NUMBER, --min-instance MIN_SUBGRAPH_INSTANCE_NUMBER
                        The minimum instance number of a subgraph, subgraph
                        with fewer instances will not be detected
  --min-nodes MIN_SUBGRAPH_NODE_NUMBER
                        The minimum node number of a subgraph, subgraph
                        instance with fewer nodes will not be detected
  --max-nodes MAX_SUBGRAPH_NODE_NUMBER
                        The maximum node number of a subgraph, subgraph
                        instance with more nodes will not be detected
  -p SUB_SUB_GRAPH_THRESHOLD_PENALTY, --penalty
SUB_SUB_GRAPH_THRESHOLD_PENALTY
                        Penalty terms on sub-sub-graph in thresholds to
                        avoid multiple level subgraphs
  --skipped-level SKIPPED_LEVEL
                        The number of the skipped top levels
  -c, --check_result    Check the result after finish calculation
  --disable_scope_boundary
                        disable the scope boundary, the subgraph instance will
                        not be restricted to a scope
  -d, --detailed_isomorphic_check
                        check the isomorphism of name scope in detail
```

- 本项目可以按照Python Package的标准进行分发和安装，具体安装方式见安装指南
- 同时也已实现命令行运行接口，具体安装方式见使用指南

5. 其他相关链接

- API文档
- 更新日志
- 测试结果
- 参考文献
- 外部依赖