开题报告中确定的论文研究计划，按计划应该完成的工作；

已完成的论文研究工作和取得的阶段性成果；

存在的问题、尚未完成的工作及后期工作的设想和安排。

**一、开题中的研究重点与研究计划**

   在已有的图计算模型中，同步模型存在“木桶效应”问题；异步模型存在大量的冗余计算；混合计算模型为实现同步与异步间自由切换产生了额外的内存与预测开销；而自适应模型中，每个计算节点通过引入限定值动态调整节点间相对进度，从而解决了同步、异步、混合模型下的问题，本文在已有的自适应计算模型的原型下，提出了新的基于消息量预测的自适应图计算模型，该模型可在慢机的情况下规避上述模型的弊端，加速并行图计算，本文的研究重点与研究计划如下。

   1）提出基于消息量预测的自适应并行图计算模型 MP-AAP（Message Predict Adapt Async Parallel），并将针对消息量的预测可转化成对运行时间及消息到达速率的预测

   2）我们对时间的预测定义成机器学习中的回归问题，我们拟采用均方相对误差作为评价指标；然后，我们选取机器学习中适合做数值型预测的回归模型作为候选，进行比较，从而分析各模型在分布式图计算时间预测问题上的优缺点，并选取模型进行预测。

   3）我们选取包含sssp、bfs、pagerank、wcc、sampling、hits、label propagation、wcc\_hashmin等工业界与学术届常用的图计算应用，并尝试将算法进行分配，针对每类应用，给出不同的特征提取方案及模型选取方案。

   4）最后我们将选取的预测模型并嵌入进自适应图计算系统，并将基于消息量预测的自适应系统与同步，异步，以及现有的自适应原型的系统进行对比，评估效果。

**二、已完成的论文研究工作和取得的阶段行成果**

**1、基于消息量预测的自适应并行图计算模型**

  MP-AAP 采用Grape并行图计算模型，我们在此进行简要的介绍

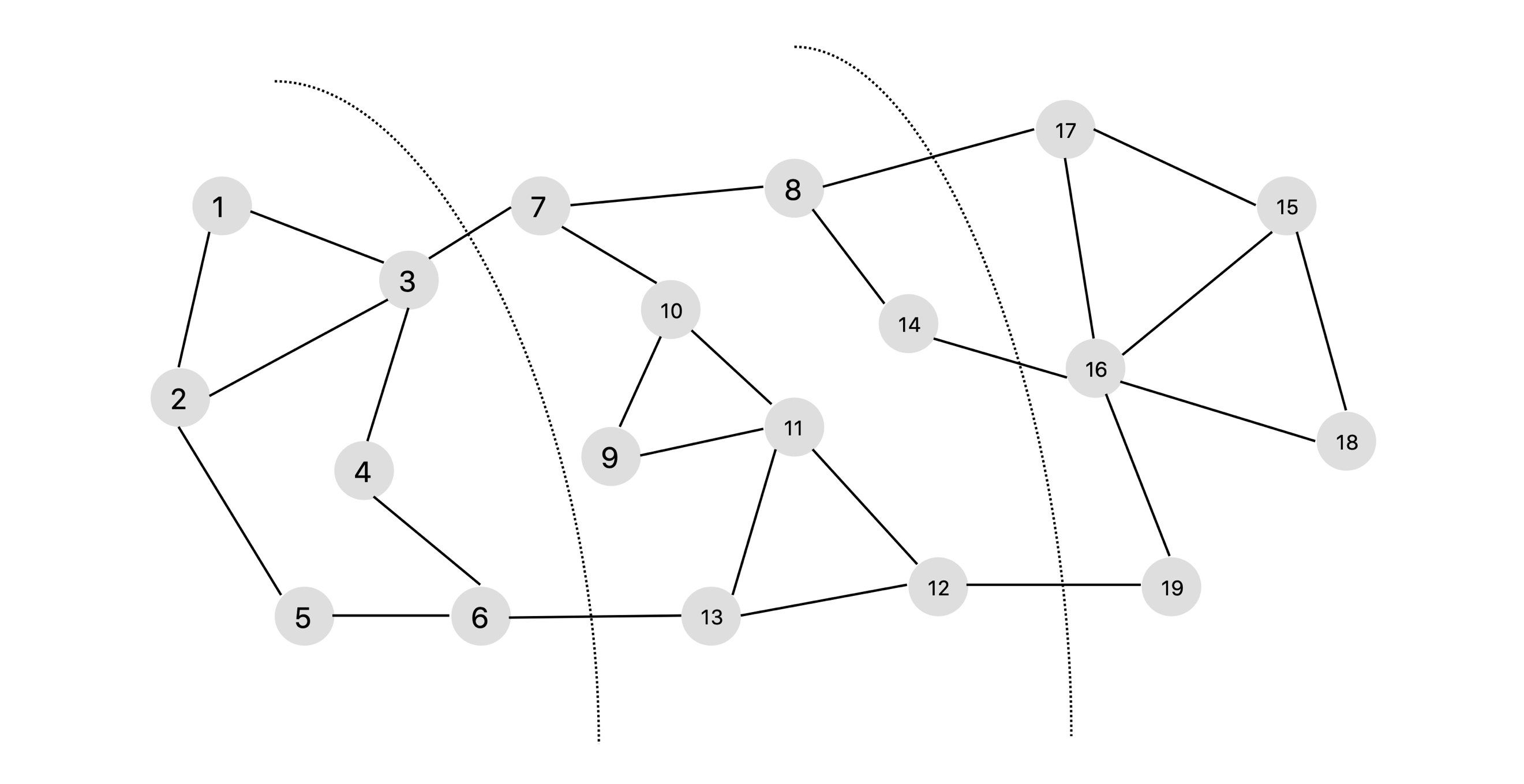
**图切分**：对于有向或无向图 G=(V,E,L)，其中V是所有点的集合，E=V\*V是所有边的集合，L代表每个点（或边）上的标签，代表属性图；给定任意一个自然数n，我们采取边分割或点分割策略将图G分割为F=(F1,.....Fn), 其中Fi是G的子图，Fi.I 代表其它子图的点指向本子图点的集合，Fi.O代表本子图的点之前其它子图的点的集合，Fi.i 并 Fi.O代表border node。

**编程模型**：图计算问题作为图查询问题的一类，与Grape编程模型类似，对于回答某个图计算问题Q，需要用户指定三个函数：

**2、算法分类与时间预测**

本章讨论了学术届与工业届常用的9种经典图应用算法，包括 sssp、bfs、pagerank、wcc、sampling、hits、label propagation、wcc\_hashmin、louvain, 我们发现在算法运行过程中，通常一次迭代不需要访问所有的内部点，因此在PIE模型下，我们调查了上述算法的运行时行为，并将行为定义为每一次超步实际访问的活跃点的集合，之后我们调查了不同算法在一次图任务计算中该集合的变化情况，最后我们将上述图算法分为三类，针对每类我们给出了不同的图特征提取方案，并采用不同学习模型进行训练预测。实验表明，我们我们预测的效果十分准确。

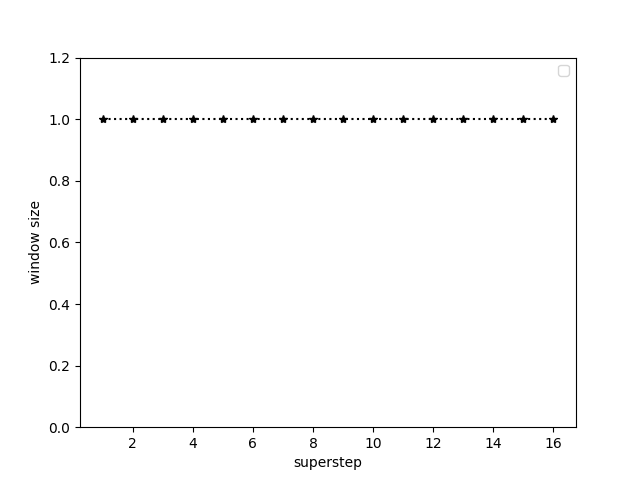
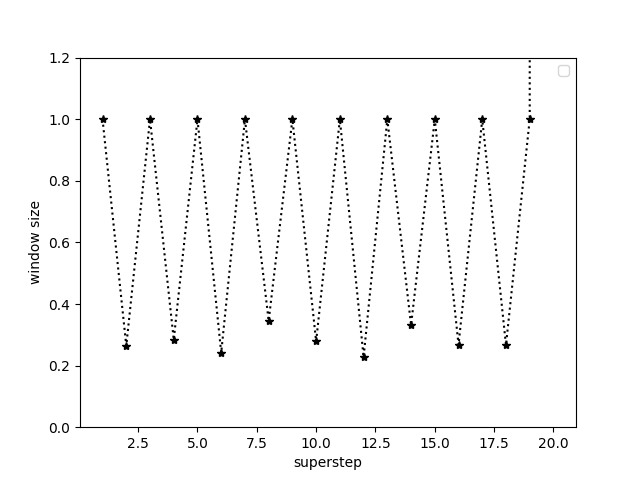
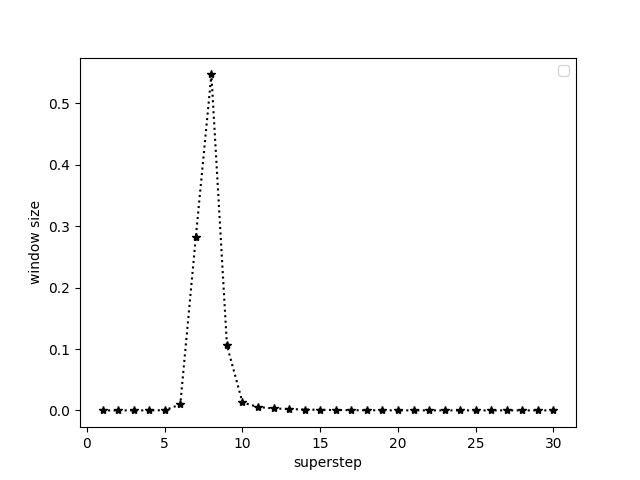
1. Active Nodes Window 行为

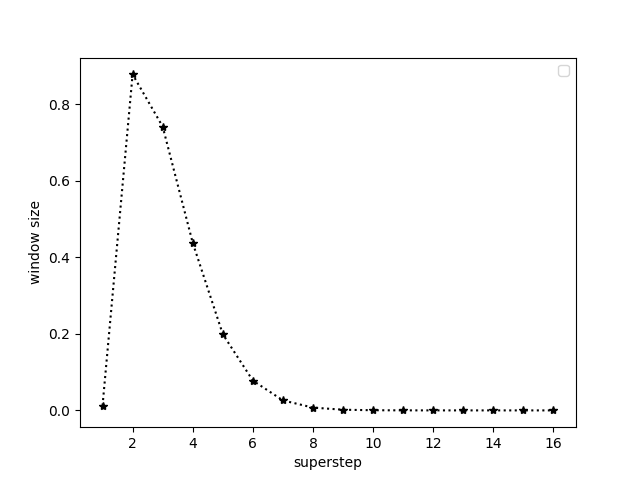
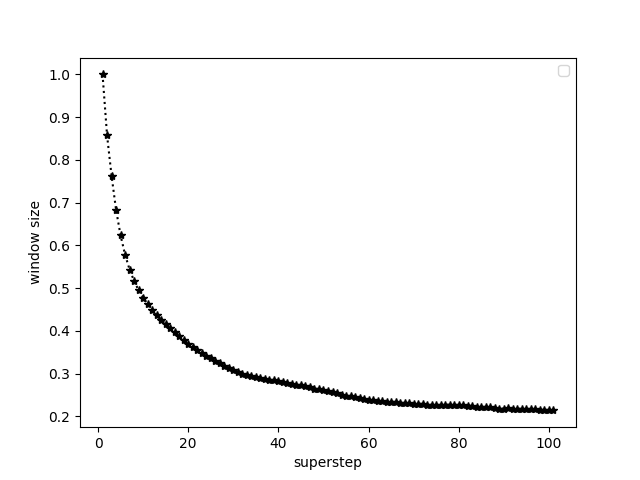
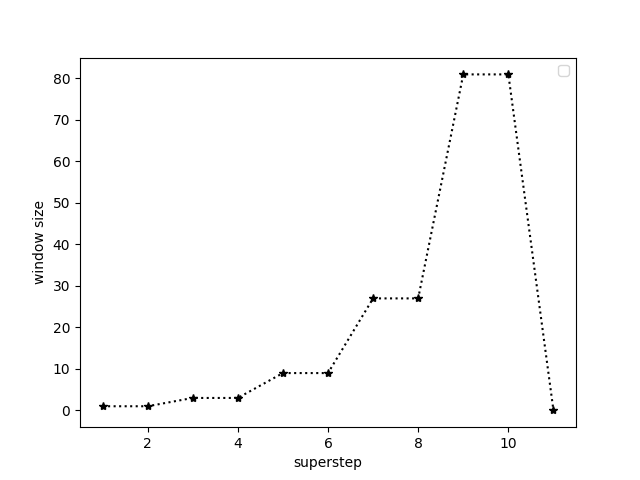


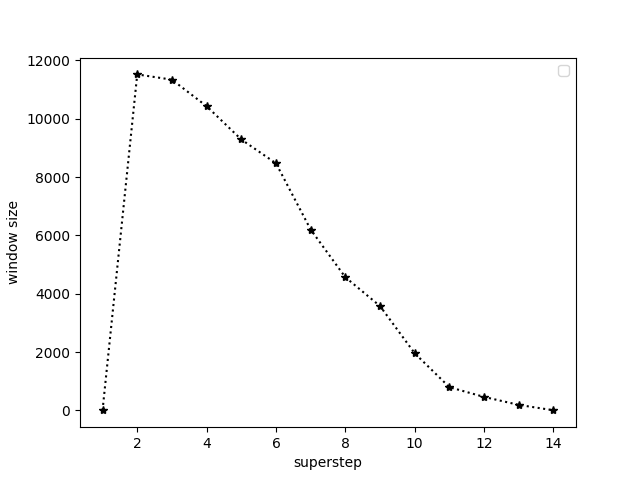
我们以小的图数据为例，介绍不同种类图算法的Active Node window的变化情况。如图所示，图包含19个节点以及26条边，假定该图采用边分割的策略分为3个子图F1，F2，F3,其中F1包含点v1={v1,v2,v3,v4,v5,v6}以及与这些点所连接的边，F2包含点V2={v7,v8,v9,10,v11,v12,v13,v14}以及与这些点所连接的边，F3包含点V3={v15,v16,v17,v18,v19}以及与这些点所连接的边其中F1与F2有两条边相连，F2与F3有3条边相连。

图2展示了pagerank，sssp以及hits算法在5个超步运行下的active node window情况，每个超步中，浅灰色的点代表非活跃点，深灰色点代表活跃点；其中图2-1展示了pagrank在前5轮超步下的working window，图中所有的点在每轮计算中均为活跃点，同样每个Fragment均参与了每轮计算；图2-2展示了sssp在前5轮超步中下的working window，可以看出5轮迭代计算的working window分别是W1={},W2={},W3={},W4={},W5={}, 并且前两轮计算，只有F1处于活跃状态，第三轮是F1，F2处于活跃状态，但是整体的active node数目很少，从4轮迭代之后，F1,F2,F3均处于活跃状态，且整体active node数目增多；图2-3代表了hits算法在前5轮超步下的working window,可以看出该算法在第一，第三，第5轮迭代中，所有点均为活跃点，而第二，第四轮迭代中只有部分点（接收更新小消息的点）处于活跃状态。

因此我们可以看出，不同的图算法在每轮迭代计算中，active nodes有不同的变化效果，接下来我们给出所有算法的现实中图数据上的active node变化折线图，所有算法都需要实现上述用户定义的PIE函数。







从图中可以看出，pagerank，label-propagation算法在每轮计算中，所有点均处于active状态，因此我们将该类算法称为Always Active Style, 而sssp bfs sampling 算法的active node window在折线图上表现为先升高，后降低（其中sampling之后采样终止时降为最低），wcc wcc-hashmin算法active node window在折线图的表现为开始升高，之后逐渐下降，我们将上述5种算法均成为Message Dependency Style ，因为每轮迭代计算活跃点的总数随着以来与当前Fragment接收到的消息数，（后面我们会给出不同种类的算法如何指导我们抽取不同的图计算运行时特征），hits 以及louvain算法active node window 表现出一定的规律性。其中hits算法在奇数轮迭代计算中，所有的点均处于active 状态，而偶数轮的迭代计算中部分点处于active状态，这类算法典型的操作一次计算流程分为多个超步，因为我们将这些算法成为Multiple Parse Style，这类在图算中也十分常见，包括MST MM等。

**Always Active Style (种类一)**

在该风格下，每个子图在每一次超步计算中，都需要遍历所有的内部点，完成规定的图计算任务，并将本轮计算生成的消息传给邻居fragment（前面要讲述一下fragment），表算法-1描述了该类算法在IncEval上的行为，其中由于PIE模型下的Peval只在图计算开始时执行一次，因此本文只考虑IncEval的内容

表

我们可以看出，输入M代表该fragment接收到来自于上一轮迭代其它fragment发来的消息，在IncEval中，首先针对接收到的消息调用聚合函数，并将消息更新到本fragment的内部点中（多少行），其次针对本fragment的所有内部点进行相应计算（多少行）最后将更新后的消息发往其他的fragment（多少行）。

作为一个例子我们给出PIE模型下pagerank的IncEval计算过程，从算法中我们清楚的看到，这看看能不能用论文的那里

**Message Dependency Style (种类二)**

在该风格下，通常计算开始时只有一个点或若干点处于活跃状态，而其它点是否参与本次计算取决于是否接到到消息，因此消息的传播条件以及决定了这类算法的活跃点数目，如果算法-4所示，开始条件下，只有一个点处于活跃状态，其余点均为非活跃点，在IncEval开始时，首先针对接收到的消息调用聚合函数，并将消息更新到本fragment的内部点中（多少行），同时记录更新的内部点，标记为active状态，接下里，针对本fragment活跃的内部点，进行相应计算，而没有更新消息的内部点仍处于非活跃状态，不参与本轮迭代计算，最终将更新后的消息发往其他的fragment。跟always对比一下。

给出sssp的例子

**Multiple Phase Style (种类三)**

如算法-5中所描述，该类图算法无法在一次超步中完成一次图任务计算，例如hits算法，该算法与pagerank算法类似，都是对一个网络图做结构分析，最早用于搜索，现在也常用来做社交网络结构分析，但pagerank算法不同的是，hits算法一个(网络)节点的重要性分成了两种属性，分别为Authority于Hub，其中Authority认为如果一个页面提供了关于某个主题的信息，那么这个页面就是有价值的，这样的页面就是authority页面，因此如果有很多其它的页面都指向该页面，说明这个页面的authority值就高；Hub任务还有一些页面，比如google页面，虽然它本身不提供任何主题信息，但是从这个页面出发，可以跳转到很多有价值的页面上去，这样的页面就会成为hubs，因此如果某个页面指向了很多有authority值高的页面，那么它的hub值就会高。在PIE计算模型中，因为一轮中，每个点既要算本身的authority值，也要算hub值，并且hub值的计算要依赖于周围邻居节点的aythority值，所以无法用PIE模型一轮的IncEval同时算出所有点的authority值与hub值，因此需要拆分成两次IncEval去做，其中一次用于计算所有的Authority值，另一次根据上轮的authority值计算hub值。通常情况下，这类算法会将一次图计算任务Pi分为PIE模型下的n次IncEval：Pi0…Pin-1，这类算法还包括louvain，mst。

我们以hits算法为例。。。

**算法特征提取与运行时间预测**

在PIE计算模型中，某个fragment（术语表）每轮的计算开销主要依赖于活跃点的数目与接收到的消息数。其中活跃的点是该fragment内部点的子集，因为在某些算法中，并不是所有的点都会被接收到的消息激活，我们将某个fragment Fi在第s次IncEval迭代计算的活跃点集合记为Wis，因此在第s次计算中，Fi的计算开销可以表示为Ks=wis + outnode +msgnum，（解释一下）

等式

上述等式也同样表达了，针对一轮计算，某个fragment的执行时间取决以下三个要素

**3、系统设计与实现**

三、存在的问题与后期工作安排