北京航空航天大学计算机学院

硕士学位论文中期检查报告

论文题目：基于消息量预测的自适应并行图计算系统性能优化研究

专　　业：计算机技术

研究方向：分布式图计算系统

研 究 生：李东泽

学　　号：ZY1706212

指导教师：樊文飞 教授

北京航空航天大学计算机学院

2019 年 08 月 30 日

目录

基于消息量预测的自适应并行图计算系统性能优化研究

1 论文的研究重点与研究计划

1.1 研究背景与意义

图计算是以“图论”为基础的对现实世界“图”结构的抽象表达。在实际的应用中，存在着许多图计算问题，如图的连通性[1]、单源最短路径[2]、网页排序[3]等，同样在社会或科学领域中的许多问题都可以通过转换成图计算模型从而得到解决，例如: Google 公司每天都需要对数亿级别的网页进行排序[4]; 生物医学上需要对蛋白质进行子图匹配，分析蛋白质间的相互作用从而开发出更有效的临床药物[5]。因此，对图数据的分析和计算具有很重要的现实意义。

早期的图计算数据规模较小，大部分使用 BGL 等单机图处理算法[6]，极大的限制了图计算问题的规模。后来随着图规模的增长，出现了 Parallel BGL[7]等图计算系统，但是系统本身不具备良好的稳定性。近些年来，随着大数据时代的来临，大图数据的分析给图计算带来了极大的挑战。于此同时，MapReduce[8]作为一个并行的计算框架，及其开源实现 Hadoop[9]为海量数据的处理提供了便利，但是由于图计算过程中通常涉及多轮迭代，因此该框架不能高效的进行并行图计算。

针对上述图计算特点与图处理中所面临的问题，面向大规模图计算的研究吸引了越来越多的研究者投身其中。最近，在计算模型和运行优化方面涌现出不少具有前瞻性的图计算系统，代表性的图系统包括 Giraph[10]、GPS[11]、GRACE[12]、GraphLab[13]、PowerSwitch[14]、GRAPE+[15]。它们在计算模型上分别属于同步、异步、混合以及自适应计算模型，它们围绕着减少迭代次数加快算法收敛、减少消息传递数目、实现负载均衡这 3 个优化目标，分别采用一种或多种优化技术对自身进行优化，但是它们往往都是在优化某个目标的同时牺牲了其它性能。

因此本课题以自适应并行图计算为框架，采用机器学习模型，针对图计算过程中接收的消息量进行预测，最终合理的动态调节各计算节点间的相对进度，从而达到性能优化。

1.2 论文的研究目标

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。

1.3 论文主要研究内容

本文研究重点与研究内容如下：

1. 提出基于消息量预测的自适应并行图计算模型，以下简称 MPAP 模型（Message Predict Async Parallel），并将对消息量的预测转化为对运行时间及消息到达速率的预测。

2. 我们将对算法运行时间的预测问题定义为机器学习中的回归问题，使用均方相对误差（MSRE）做为损失函数，最终选取机器学习中适合图计算领域的回归模型作为候选，进行训练预测，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。

3. 我们选取包含单源最短路径（SSSP）、深度优先搜索（BFS）、网页排序（PageRank）、节点结构分析（HITS）、图采样（Graph Sampling）、图连通性（WCC WCC-HASHMIN）、标签传播（Label Propagation）、社区发现算法（Louvain）等 9 种学术届与工业界常用的图计算应用算法，并尝试将算法进行分类，针对每类算法，给出不同的特征提取方案，并展示预测效果。

4. 我们在已有的自适应图计算系统原型下，开发完成基于消息量预测的自适应并行图计算系统，并与同步、异步、及自适应原型的系统进行对比，评估效果。

1.4 关键技术与难点

1. **图应用算法的分类与特征选取**：收集运行时信息的过程及机器学习领域特征提取的过程，特征提取结果的好坏直接影响接下来模型的训练与预测，图计

算中的特征提取涉及多方面因素，包括图结构本身，如图的规模、平均度数、各子图边界点个数；运行时信息，如消息的接收数量与质量等不同特征。本课题从分布式图计算角度出发，会对上述算法从消息传递、本地计算以及图数据特性等不同方向分析，从而将图算法进行分类，给出每一类算法的特征选取方向与不同模型的预测效果对比。

2. **模型的选取**：虽然我们可以将运行时间的预测定义成机器学习中的回归问题，但存在众多的回归模型需要我们进行分析，如局部加权线性回归模型、岭回归模型、树回归模型甚至使用神经网络模型，不同模型在训练的时间及效果上差异很大，因此结合图计算领域约束，我们期望找出每类算法适合的回归模型，并作出训练与预测。

3. **图计算系统实现**：最终，我们需要将训练好的模型嵌入分布式图计算系统中，嵌入过程主要包括运行时日志采集、特征提取、模型训练及预测，因此整个过程涉及分布式图计算系统的图存储模块、消息传递模块、及与模型的训练预测有关的自适应调整模块的开发。

1.5 论文预期成果形式

1. 给出基于消息量预测的 MPAP 模型。

2. 针对多种工业界与学术界常用的图应用算法，在 PIE 模型下，找到不同算法的运行时特点，并对算法进行分类，针对每一类算法给出对应特征提取方案与不用模型的训练效果对比。

3. 基于已有的自适应图计算系统，开发基于消息量预测的自适应并行图计算系统，并与同步、异步、自适应原型模型进行实验对比，期望运行时间较同步、异步、自适应原型系统相比，有一定的缩短，消息量减少。

4. 基于后端图计算服务，开发 web 页面，对整个消息量预测的图计算流程进行功能性展示。

2 已完成的工作与阶段性成果

2.1 基于消息量预测的自适应并行模型

2.1.1 PIE 编程模型

由于 MPAP 采用与 GRAPE+相同的编程模型，即 PIE 模型，因此本节首先简要介绍一下该模型。

表 2-1 术语表

fragment	图分割后的子图
inner node	子图的内部点
border node	边界点，这些点与其它 fragment 相连
outer node	不属于本 fragment，但与本 fragment 中的点直接相连的点
PEval	PIE 模型的估值函数，即 P
IncEval	PIE 模型的增量计算函数，即 I
Assemble	PIE 模型的汇集函数，即 E
MPAP	本文提出的基于消息量预测的异步模型，Message Predict Async Parallel
worker	物理机器或虚拟机器，fragment 存储在某个 worker 上
coordinator	也是 worker，只是用于与各个 worker 通信，控制整体运行状态，参与模型收集与训练工作

图符号术语：给定某有向图或无向图 $\mathrm{G}=(V,E,L)$，其中 V 代表图中点的有限集合，$\mathrm{E}=V\times V$ 代表图中边的集合，L 代表图中顶点或边上的属性；

给定某一自然数 m，我们采用某种分图策略 ω 将图 G 分割为 m 个片段 $\mathrm{F}=(\mathrm{F}_1,\dots\dots,\mathrm{F}_m)$，其中每个 $\mathrm{F}_i=(\mathrm{V}_i,\mathrm{E}_i,\mathrm{L}_i)$ 均为 G 的子图，且有 $\mathrm{V}=\bigcup_{i\in[1,m]}V_i$，$\mathrm{E}=\bigcup_{i\in[1,m]}E_i$，$\mathrm{V}_i\in V$，$\mathrm{E}_i\in E$，针对每个顶点 $v\in\mathrm{V}_i$，有 $\mathrm{L}_i(v)=\mathrm{L}(v)$，针对每条边 $\mathrm{e}\in\mathrm{E}_i$，有 $\mathrm{L}_i(\mathrm{e})=\mathrm{L}(\mathrm{e})$。

MPAP 模型采用边分割或点分割策略的来分割图 G，当使用边分割时:

(1) $\mathrm{F}_i.I$ 代表点的集合 $v\in\mathrm{V}_i$，其中针对每个点存在一条边 (v',v)，使得

$v' \in F_j$，且(i ≠ j)。

(2) $F_i.O$ 代表点的集合 $v \in V_i$，其中针对每个点存在一条边 (v, v')，使得 $v' \in F_j$，且(i ≠ j)。

我们将上述 $F_i.I \cup F_i.O$ 成为子图 F_i 的边界点(border node)集合，本文用到上的术语见表 2-1.

PIE 编程模型：众所周知，图计算问题作为图查询问题的一类，针对某一问题实例给定问题 Q，为回答该问题，MPAP 模型需要用户提供以下三个函数：

1. PEval (局部估值函数): 该函数将子图 F_i与问题 Q 作为输入，并在子图 F_i 上计算回答该问题 Q，得到结果 $Q(F_i)$，其中针对该问题 Q 的所有单机图算法均可直接使用。

2. IncEval (迭代增量函数): 该函数将已有结果 $Q(F_i)$、子图增量 ΔF_i、与问题 Q 作为输入，并在更新后的子图 $(F_i \oplus \Delta F_i)$ 上回答问题 Q，保证有 $Q(F_i \oplus \Delta F_i) = Q(F_i) \oplus \Delta O$，其中 ΔO 是在原有 $Q(F_i)$ 的基础上做的更新计算。

3. Assemble (聚合函数): 该函数用于聚合其他子图 F_j 在 PEval 或 IncEval 阶段计算所产生并同步本子图 F_i 的更新量，最后将聚合结果更新到已有结果 $Q(F_i)$。

2.1.2 MPAP 模型

本节我们介绍基于消息量预测的并行图计算模型，即 MPAP 模型。

初始配置：MPAP 模型本质上仍然采用上节介绍的 GRAPE PIE 编程模型，给定某一图计算问题实例及在该实例上的问题 Q，MPAP 模型需要用户指定上述 PIE 函数。首先，将输入的图数据 G 通过边分割或点分割策略分为多个片段 $F = (F_1, \ldots\ldots, F_m)$，每个片段 F_i 存储在某台虚拟机 $\mathcal{W}_i$ $(i \in [1, n])$上，除此之外，MPAP 模型需要 $\mathcal{W}_0$ 做为 coordinator 节点，用于完成后续将要提到的集群终止状态检测与运行时特征的收集、训练和预测任务。因此，若 $n \leq m$，则多个片段 F_i 会分到同一个虚拟机上并分享内存。

并行模型：与 AAP 模型不同的是，MPAP 模型还需增加以下声明，其中所有在 PEval 中的声明均在 IncEval 中共享。

1. 状态变量：PEval 中每个片段 F_i 都会声明并维护一个集合 $\mathcal{C}_i$，该集合用于存储片段中每个点的状态信息，其中包括结果信息，在接下来的 IncEval 阶段中，该集合也用于更新有用的消息。

2. 聚合函数：PEval 中需要指定聚合函数来解决多台机器传递的消息同时作用于相同的变量的问题，常用的聚合函数如 sum 函数、max 函数、min函数等。

3. 特征提取：IncEval 中每个片段 F_i 在接收处理本轮的消息后，会向 coordinator 节点发送本轮的运行时日志，该日志主要包含当前该片段的状态信息与特征信息。

表 2-2 MPAP SSSP PEval

Input: $Fragment\ \mathrm{F}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathrm{L}_i), source\ vertex\ v$

Output: $A\ set\ of\ Q(\mathrm{F}_i)\ consisting\ of\ current\ dist(v, v')\ for\ all\ v' \in \mathrm{V}_i$

$MPAP\ Status\ Variables\ \mathcal{C}_i$:

$\mathcal{C}_i.dist$: $for\ each\ node\ s \in \mathrm{V}_i, a\ double\ variable\ representive\ dist(v, s)$

$\mathcal{C}_i.visited$: $for\ each\ node\ s \in \mathrm{V}_i, a\ boolean\ variable\ representive\ visited\ or\ not$

$MPAP\ Aggregation\ Funtion$: $\min dist(v, v')$

$MPAP\ PEval$

1. $initialize\ Status\ Variable\ \mathcal{C}_i.dist\ with\ \infty, representive\ max\ value\ of\ distance$
2. $initialize\ Status\ Variable\ \mathcal{C}_i.visited\ with\ false$
3. $initialize\ empty\ priority\ queue\ Q, set\ dist(v, v) \coloneqq 0$
4. $Q.add(v, dist(v, v))$
5. $While\ Q\ is\ not\ empty\ do$
6. $u \coloneqq Q.pop()\ //\ with\ priority\ queue, here\ pop\ value\ is\ minimal\ distance$
7. $set\ \mathcal{C}_i.visited(u) \coloneqq true$
8. $for\ each\ child\ u' of\ u\ do\ //\ only\ with\ u'\ has\ not\ been\ visited$
9. $d \coloneqq \mathcal{C}_i.dist(v, u) + \mathrm{L}_i(u, u')$
10. $if\ d < \mathcal{C}_i.dist(v, u')\ then$
11. $\mathcal{C}_i.dist(v, u') \coloneqq d$
12. $if\ u' is\ inner\ vertex\ then\ Q.add(u', \mathcal{C}_i.dist(v, u'))$
13. $Q(\mathrm{F}_i) \coloneqq \{(v', \mathcal{C}_i.dist(v, v')) \mid v' \in \mathrm{V}_i\}$
14. $send\ message$: $M_i \coloneqq \{(v', \mathcal{C}_i.dist(v, v'), 1) \mid v' \in \mathrm{F}_i.O\}$

下面我们以单源最短路径问题为例，给出对应的 MPAP 并行模型

单源最短路径问题中，给定一个带权有向图 $\mathrm{G} = (V, E, L)$，其中 L 代表每条边上的权重且$L \in \mathbb{R}$，给定某点 v 做为源点，现要计算从源点 v 到图中其它各顶点的最短路径长度，该长度即为 v 到目标顶点 v' 的路径中所走边的权重之和，该

算法输入输出表述如下:

Input: 带权有向图 $G=(V,E,L)$，与图 G 中某顶点 v

Output: $distance(v,s)$ $s \in V$

如表 2-2 所示，MPAP 在 PEval 阶段中采用了众所周知的求单源最短路径的算法- Dijstra 算法。但与该算法单机版本不同的是 1) 我们针对该 fragment 中的每个点定义元组 $(dist, visited)$，其中 $dist$ 代表该点与原点的距离，初始化为无穷大，意味着与原点不可达；$visited$ 代表在本轮计算中该点是否被访问过，初始化为 false 。该元组声明在 PEval 中，但也在 IncEval 中共享，后续也同样用于更新 IncEval 接受来自其它 worker 的消息。2) 我们采用$\min(v,v')$ 做为聚合函数，即当该 fragment 中某个点接收到了来自多个 worker 的更新消息，我们选取其中的最小值做为最终的唯一更新消息。

上述 PEval 过程执行结束后，每个 fragment 会将外部点的距离发送到对应的 worker 上，MPAP 消息传递的过程下一节详细阐述。

IncEaval 的过程如表 2-3 所示，首先对接收的消息执行聚合函数，并更新对应点的$dist$ 值，之后同样采用 Dijstra 算法计算本轮最短路径，并将计算后的最新值以消息的方式传递给其它 worker。与 AAP 模型不同的是，IncEval 过程还需将本轮计算的运行时信息发送给 coordinator 所在的节点，用于接下来的模型训练与预测，详细内容下节阐述。

如表 2-4 所示，除 PEval 与 IncEval 函数外，MPAP 模型还需要 Attl 函数，用于动态调整各 fragment 间相对进程，该函数对用户透明，具体细节后续给出。此例中不需要 Assemble 函数，因此不做介绍。

消息传递：每轮计算结束后，每个 worker $\mathcal{W}_i$ 都会收集属于自己的 fragment 的更新结果，这些结果来自与每个 fragment 内部更新数据的改变量，并将更新结果发送给其他的 worker $\mathcal{W}_j$，为实现该消息传递机制，每个 worker 需声明维护以下数据结构：

(1) 图中全部点与该点所属 worker 的映射，以便顺利将该点的更新消息发送给对应 worker

(2) 一个缓冲 buffer，用于接收来自其他 worker 发送的消息

MPAP 的消息传递过与经典 AAP 模型相同，是点对点通信、且发生在图计算的任意时刻，即任意 worker 在任意时刻都可将消息发送给 worker $\mathcal{W}_j$而不需要关系 $\mathcal{W}_j$ 此时处于状态，同样，$\mathcal{W}_i$ 也会在任意时刻接受来自其他 worker 的

消息，保存在缓冲 buffer 中，该过程不会阻塞任何计算过程。但与 AAP 模型不同的是，每个 fragment 除发送本轮更新的消息外，还需像 coordicator 节点发送本轮的运行时信息，该信息用于后续的模型训练。

表 2-3 MPAP SSSP IncEval

$Input: Fragment\ \mathrm{F}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathrm{L}_i), partial\ result\ Q(\mathrm{F}_i), received\ message\ M_i$
$Output: new\ result\ Q(\mathrm{F}_i \oplus M_i)$
$MPAP\ Status\ Variables\ \mathcal{C}_i$:
$\mathcal{C}_i.dist: for\ each\ node\ s \in \mathrm{V}_i, a\ double\ variable\ representive\ dist(v,s)$
$\mathcal{C}_i.visited: for\ each\ node\ s \in \mathrm{V}_i, a\ boolean\ variable\ representive\ visited\ or\ not$
$MPAP\ Aggregation\ Funtion: \min dist(v,v')$
$MPAP\ IncEval$
1. $initialize\ empty\ priority\ queue\ Q$
2. $new\ M_i \coloneqq aggregation\ function\ (Mi)$
3. $for\ each\ dist(v,v')in\ new\ M_i$ do （加）
4. $Q.add(v, dist(v,v'))$
5. $While\ Q\ is\ not\ empty\ do$
6. $u \coloneqq Q.pop()\ //with\ priority\ queue, here\ pop\ value\ is\ minimal\ distance$
7. $set\ \mathcal{C}_i.visited(u) \coloneqq true$
8. $for\ each\ child\ u' of\ u\ do\ //only\ with\ u'\ has\ not\ been\ visited$
9. $d \coloneqq \mathcal{C}_i.dist(v,u) + \mathrm{L}_i(u,u')$
10. $if\ d < \mathcal{C}_i.dist(v,u')\ then$
11. $\mathcal{C}_i.dist(v,u') \coloneqq d$
12. $if\ u' is\ inner\ vertex\ then\ Q.add(u', \mathcal{C}_i.dist(v,u'))$
13. $Q(\mathrm{F}_i) \coloneqq \{(v', \mathcal{C}_i.dist(v,v')) \mid v' \in \mathrm{V}_i\}$
14. $send\ message: M_i \coloneqq \{(v', \mathcal{C}_i.dist(v,v'), step) \mid v' \in \mathrm{F}_i.O\}$
15. $send\ runtime\ log\ info: R_i \coloneqq$
$\{(fid, ivnum, ovnum, tvnum, edge\ num, message\ embedding)\}$

表 2-4 MPAP SSSP ATTL

Input: $Fragment\ \mathrm{F}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathrm{L}_i), current\ received\ message\ M_i, current\ step, \max step, min\ step$

Output: $waiting\ time\ (s)$

$MPAP\ Attl$

1. $predict_{time} \coloneqq getPredictTime(feature\ vector)$
2. $predict_{msgrate} \coloneqq getPredictMsgRate(feature\ vector, step)$
3. $if\ predict_{time} * predict_{msgrate} > number\ of\ M_i\ then$
4. $predict_{msg} \coloneqq predict_{time} * predict_{msgrate}$
5. $else$
6. $predict_{msg} \coloneqq \max(M_L, number\ of M_i) + \Delta\tau * predict_{time} * predict_{msgrate}$
7. $attl \coloneqq (predict_{msg} - number\ of\ M_i) / predict\_msgrate$
8. $return\ attl - idle_{laststep}$

MPAP 模型：MPAP 模型中，为解决同步模型下的“木桶效应”及异步模型下的冗余计算问题，每个计算节点 Pi 引入了限定值DS_i $(Delay\ Stretch)$，之后每轮迭代计算开始前，各计算节点需判断是否等待 DS_i 长的时间以积累更多的消息，我们给出 DS_i 基于下述函数进行动态调整：

$$DS_i = \begin{cases} (P_t * P_s - \eta_i) / P_s - T_{idle}^i & (P_t * P_s) > \eta_i \\ (\eta_i + \Delta t * P_t * P_s) / P_s - T_{idle}^i & (P_t * P_s) \leq \eta_i \end{cases}$$

上述函数中的各参数描述如下，DS_i 表示某一 fragment 新一轮计算等待的时间：

(1) η_i：表示计算节点当前接收的消息量。直观上来看，如果 η_i 越大，则当前计算节点接收消息越多，应尽快开始下轮迭代计算。

(2) P_t P_s：预测的时间与消息到达速率。

(3) T_{idle}^i Δt：T_{idle}^i 表示 fragment 前一轮计算后的空闲时间，以防止无限期的等待，Δt 超参数 处于(0,1)之间，用于控制 fragment 等待的时间，Δt 越大，则每个 fragment 等待的时间越长。

终止条件：MPAP 模型的终止条件与经典 AAP 模型相同，即新一轮迭代计算开始前，如果当前没有任何接收到的消息，则会像 coordinator 发送结束命令，当 coordinator 接收到所有 worker 的命令后，会广播各个 worker 终止命令，各个 worker 会对此返回 ack 确认自己是否真的结束，如果某个 worker 仍有任务需要

计算，则回应 wait 命令，继续进行下一次计算，coordicator 重新进入监听等待状态。

2.2 算法的分类

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。

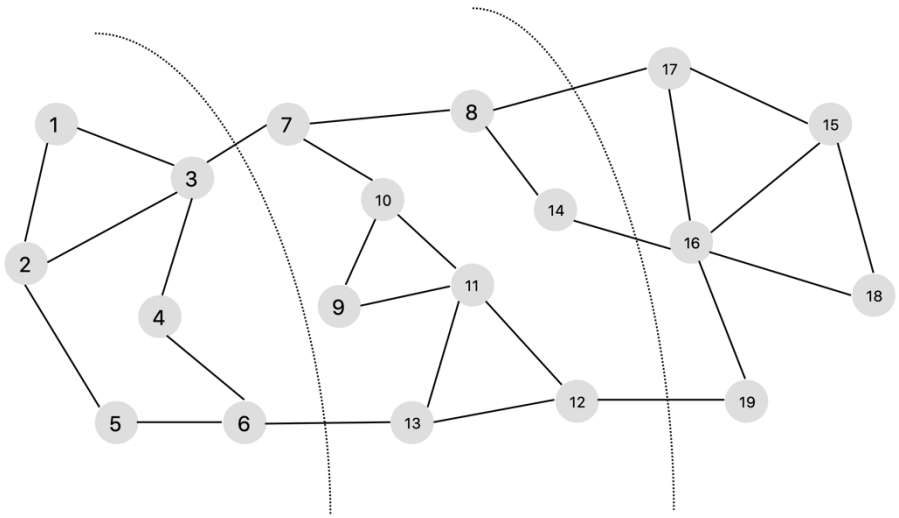
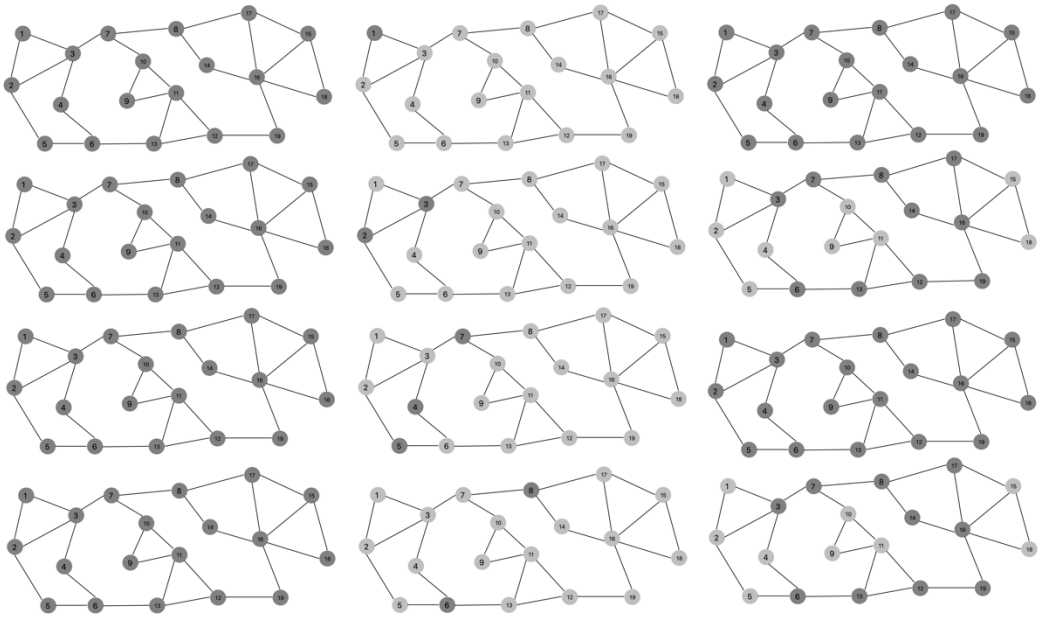


图 2-1 样例图

2.2.1 Active Nodes Window 行为

我们以图 2-1 为例，介绍同种类图算法的 Active Node Window 的变化情况。如图所示，图包含 19 个节点以及 27 条边，假定该图采用边分割的策略分为 3 个子图 F_1， F_2， F_3,其中 F_1包含点 $V_1 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$以及与这些点所连接的边， F_2 包含点 $V_2 = \{v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\}$以及与这些点所连

接的边， F_3 包含点 $V_3 = \{v_{15}, v_{16}, v_{17}, v_{18}, v_{19}\}$以及与这些点所连接的边。其中 F_1与 F_2有两条边相连， F_2 与 F_3 有 3 条边相连。

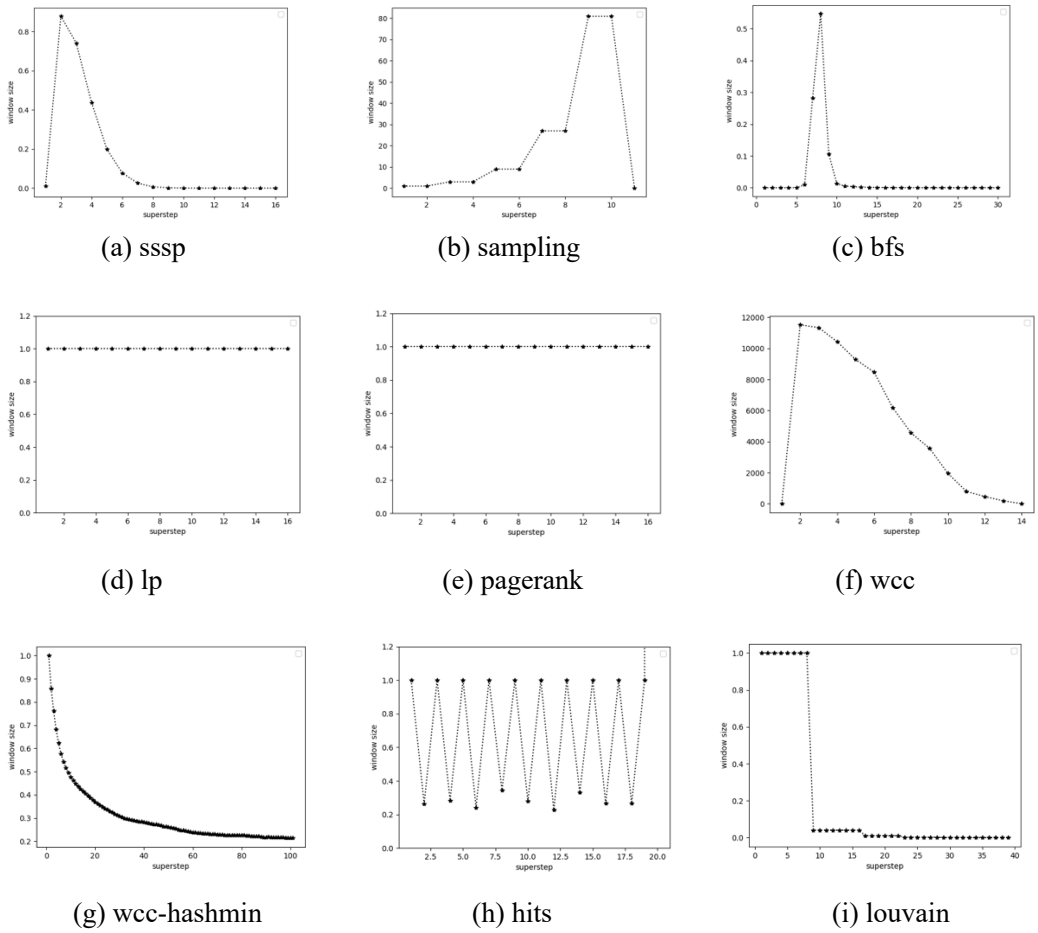


(a) pagerank (b) sssp (c) hits

图 2-2 算法 Active Node Window 行为分析

图 2-2 展示了 pagerank，sssp 以及 hits 算法在 4 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 2-2-(a)展示了 pagerank 在前 4 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-(b)展示了 sssp 在前 4 轮超步中下的 working window，分别是 $w_1 = \{v_1\}, w_2 = \{v_2, v_3\}, w_3 = \{v_4, v_5, v_7\}, w_4 = \{v_6, v_8\}$,每轮计算的活跃点数不恒定；图 2-2-(c)代表了 hits 算法在前 4 轮超步下的 working window,可以看出该算法在第一，第三轮迭代中，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 nodes 变化折线图，所有算法都需要实现上述用户定义的 PIE 函数。



(a) sssp (b) sampling (c) bfs

(d) lp (e) pagerank (f) wcc

(g) wcc-hashmin (h) hits (i) louvain

图 2-3 算法 working window 行为

从图 2-3 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 算法的折线图上 working window 表现为先升高，后降低(其中 sampling 算法在采样终止时降为最低)；而 wcc、wcc-hashmin 算法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依赖于当前 fragment 接收到的消息数(后面我们针对不同种类的图算法给不同的特征提取方案)；hits 以及 louvain 算法在折线图上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，这类算法典型的将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，这类算法在图计算中也十分常见，包括最小生成树 MST 等。

Always Active Style (种类一)

该类算法中，每个 fragment 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。

表 2-5 Always Active Style IncEval

Input: $Fragment\ \mathrm{F}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathrm{L}_i), partial\ result\ Q(\mathrm{F}_i), received\ message\ M_i$

Output: $new\ result\ Q(\mathrm{F}_i \oplus M_i)$

1. $for\ each\ v\ (v \in V_i)\ do$
2. $\quad call\ local\ compute, change\ value\ on\ v\ if\ necessary$
3. $send\ message$

我们可以看出，输入 M_i 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，该类算法对本 fragment 的所有内部点进行相应计算（line 2）最后将更新后的消息发往其他的 fragment（line 3）。

Message Dependency Style (种类二)

表 2-6 Message Dependency Style IncEval

Input: $Fragment\ \mathrm{F}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathrm{L}_i), partial\ result\ Q(\mathrm{F}_i), received\ message\ M_i$

Output: $new\ result\ Q(\mathrm{F}_i \oplus M_i)$

1. $for\ each\ v\ (v \in V_i)\ do$
2. $\quad if\ v\ is\ updated\ then$
3. $\quad\quad call\ local\ compute, change\ value\ on\ v\ if\ necessary$
4. $send\ message$

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决定了这类算法的活跃点数目，如表 2-6 所示。

Multiple Phase Style (种类三)

表 2-7 Multiple **Phase Style IncEval**

$Input: Fragment\ \mathrm{F}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathrm{L}_i), partial\ result\ Q(\mathrm{F}_i), received\ message\ M_i$
$Output: new\ result\ Q(\mathrm{F}_i \oplus M_i)$
1. $switch.getSuperStep\ \%\ k\ do$
2. $case\ i$
3. $call\ either\ Always\ Active\ Style\ or\ Message\ Dependency\ Style$
4. $send\ message$

如表 2-7 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 认为还有一些页面，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 P_i 分为 PIE 模型下的 n 次 IncEval 计算$P_{i0}, P_{i1}, \ldots, P_{n-1}$，这类算法还包括 Louvain，MST。

2.3 运行时间预测

在 PIE 计算模型中，某个 fragment 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 F_i 在第 s 次 IncEval 迭代计算的活跃点集合记为 W_{is}，因此在第 s 次计算中，F_i 的计算开销可以表示为

$$K_s = W_{is} + Num_{outnode} + Num_{msg}$$

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 1) 活跃点数目，通常代表要执行点计算的次数 2) 外部点数目，影响消息发送的数目与数量 3) 消息接收数目，通常影响活跃点数目。

运行时预测：本节我们对上述分类的算法在运行时间与消息到达速率方面进行预测，并展示预测结果。

我们将特征向量表示成$X_j = [x_1, x_2, \dots, x_m]$，即 m+1 维向量，对每个特征向量$X$ 我们都有一个运行时间 t，因此点对$d(X, t)$代表一个样本，整个数据集合为 $D = [d_1, d_2, \dots, d_n]$。该特征提取于 coordinator 节点所收集的运行时信息。下面我们使用随机森林、岭回归、局部加权回归、神经网络等方法，分别对上述算法进行训练预测，我们采用 k $(k = 10)$交叉验证的方法，使用均方相对误差做为损失函数，公式如下：

$$MSRE = \frac{\sum_{i=0}^{n} \left(\frac{(t_i - t_i')}{t_i}\right)^2}{n}$$

我们之所以采用均方相对误差做为损失函数，是因为相对于均方绝对误差与均方误差而言，均方相对误通过将误差除以实际运行时间来降低慢任务的影响。例如，现有三个真实 running time 分别是 20ms，50ms，120ms，预测的后的时间对应为 15ms，40ms, 100ms。那么如果采用均方误差来衡量模型的性能，得到的结果分别是 25，100，400，整个模型的 MSE 为 171，可以看出，120ms 的贡献了误差的绝大部份，相反，如果我们使用均方相对误差，得到的结果分别是 0.25，0.2，0.17，数据之间的差别很小，因此我们可以更好的捕捉短时间的训练样本。

第一类算法运行时间预测：

特征提取：我们从上节分析可知，该类算法 working window size 恒定，永远等于内部点数目，不会受接收的消息所影响，因此针对此类算法，我们更多的针

对子图本分进行特征提取，包括图的拓扑结构，提取的特征如下

表 2-8 第一类算法特征

fragment id	ivnum	tvnum	ovnum	edge num	msg num	graph density	proportion of node

上述特征均可从 IncEval 每轮计算中直接得到，因为图中点数目与边数目等信息很难表示图的拓扑结构，因此我们使用中 fragmentID 来表示，以达到线上训练的有效性与效率性，同样也可以使用 embedding 等方法将图的拓扑信息表示成向量（这种方式下适合离线训练），从图 2-4 中，我们随机 50 个测试数据，在 random forest 模型下，给出预测效果，可以看到我们的预测效果非常好。

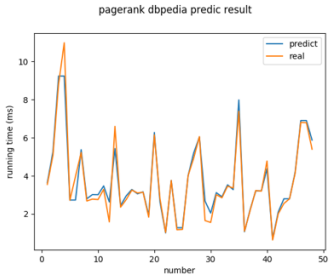


图 2-4 pagerank 运行时间预测

第二类算法运行时间预测：

特征提取：与第一类算法不同，该类算法 worker window 并不恒定，且受接收的消息所影响，因此除包含第一类算法所提取的特征外，我们还需对接收的消息做映射。提取的特征如下：

表 2-9 第二类算法特征

fragment id	ivnum	tvnum	ovnum	edge num	msg num	graph density	proportion of node	message embedding

其中 msg_embedding 是包含消息映射的向量，表示成$Z = [z_1, z_2, \ldots, z_n]$，$n$ 为超参数，代表向量的大小，z_i 代表接收消息的点 id 所映射的区间，不同的点映射到相同的区间，则采用累加的方式进行处理,如果，n 很大，则会提高模型的预测精度，但会增加模型训练的时间，相反，减小 n 则会极大缩短模型的训练时间，但会降低模型的预测精度。

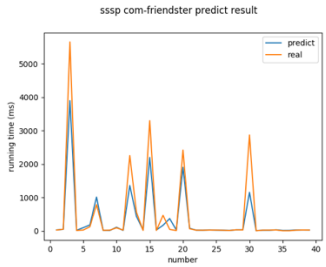


图 2-5 sssp 运行时间预测

第三类算法运行时间预测：

特征提取：与前两类算法不同的是，该类算法属于多阶段类型，且某一阶段或属于第一种类型或属于第二种类型，但上节分析得出，即便处于多阶段类型，但若干个阶段之间有相似的重复的规律，因此除上述提起的特征外，我们提取处理后轮数做为特征：

表 2-10 第三类算法特征

fragment id	ivnum	tvnum	ovnum	edge num	msg num	graph density	proportion of node	message embedding	step

消息到达速率的预测：我们采用“最近的平均预测不远的将来”思想来预测消息到达速率，即我们计算 τ 时间内所收集的消息数量m，并用 m/τ 来表示下一轮的消息达到速率。之所以使用这种方式，主要原因如下：1）我们发现，某个 fragment 接收消息的速率，不仅跟与之相连的其他 fragment 的个数有关，还与其他 fragment 中前一轮计算的活跃点集合相关，甚至还与当时网络延迟等多方面不可控因素相关，因此无法采用机器学习的方式提取特征，从而达到想训练时间那样训练消息到达速率，2）我们发现，MPAP 模型下消息的到达速率随时间 t 的减小为趋于连续平稳变化，几乎没有凸起值，因此这种最近的平均预测不久的将来思想确实有效，同样我们也用实验的方式验证了其有效性 3）即便有很多已有的模型如 RNN 模型，可以很精确的预测，但这些模型有很昂贵的部署代价，相反，我们的做法代价极低。

2.4 实验

2.4.1 系统的设计与实现

我们在 GRAPE+的基础上开发了支持 MPAP 模型的 GRAPE++。

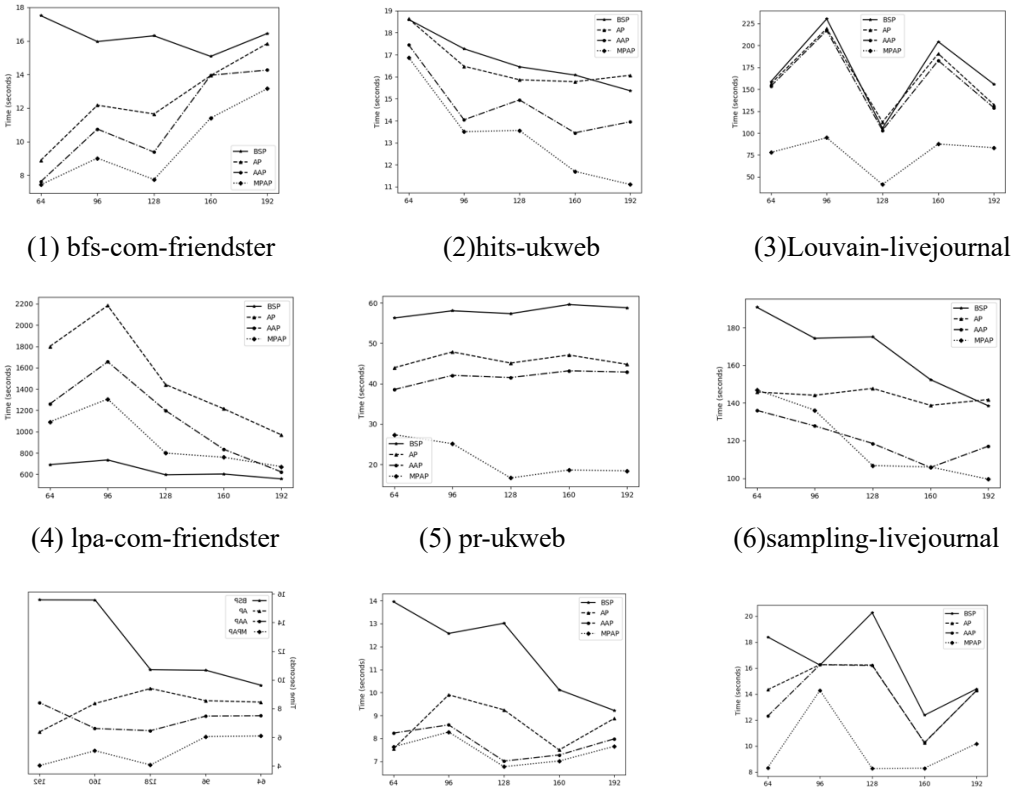
GRAPE++的架构如图所示，其中最上层用户结构层仍然采用与 GRAPE+相同的架构，即用户需指定注册 PIE 函数，中间层是 GRAPE++的核心部分，包括 MPI 控制模块，消息传递机制，MPAP 动态调整模块、运行时信息收集模块、负载均衡模块，与 GRAPE+的不同点如下：

1. MPAP 动态调整模块：为了动态调整各 fragment 间的相对进度，MPAP 模块根据每个 fragment 发送的运行时信息返回一个时间 t，用于指导各 fragment 是否等待一段时间以接收更多的消息。该模块基于对已有样本的训练所得到的模型。还包含常用的 RPC 框架，如 GRPC、Thrift 等，用于扩展该模块功能。

2. 运行时信息收集模块：每个 fragment 在每轮 IncEval 计算时都会将本轮计算的运行时信息发送给 coordinator 节点，用于指导后续模型的训练，运行时信息包括本轮运行的时间、本 fragment 的拓扑结构以及当前轮数等信息。

2.4.2 MAPA 模型性能评估

下面我们给出上述算法在 MPAP、AAP、BSP、AP 模型下的性能对比：



(1) bfs-com-friendster (2)hits-ukweb (3)Louvain-livejournal

(4) lpa-com-friendster (5) pr-ukweb (6)sampling-livejournal

(7) sssp-usa-road (8) wcc-hashmin-ukweb. (9) wcc-ukweb

图 2-6 性能对比

BFS: 如图 2-6(1)所示，在 BFS 算法以及 com-friendster 图数据上，MPAP 模型分别在 64-fragment、96-fragmnet、128-fragment、160-fragment、192-fragment 比 BSP 模型快 57%、43%、52%、24%、19%；比 AP 模型快 16%、24%、36%、18%、16%；比 AAP 模型快 2%、16%、17%、18%、7%；可以看到 MPAP 模型比 BSP 于 AP 模型具有较大的性能优势，与 AAP 原型相比，也具有一定的性能优势，但在 64 分区下优势不明显。

HITS: 如图 2-6(2)所示，在 hits 算法以及 ukweb 图数据上，MPAP 模型分别在 64-fragment、96-fragmnet、128-fragment、160-fragment、192-fragment 比 BSP 模型快 9.28%、21.8%、17.57%、27.22%、27.73%；比 AP 模型快 9.4%、18%、14.5%、25.8%、30.9%；比 AAP 模型快 3.3%、3.84%、9.28%、13%、20.4%；可以看到 MPAP 模型比 BSP 于 AP 模型具有较大的性能优势，与 AAP 原型相比，也具有一定的性能优势。

Sampling: 如图 2-6(6)所示，在 Sampling 算法以及 livejournal 图数据上，MPAP 模型分别在 64-fragment、96-fragmnet、128-fragment、160-fragment、192-fragment 比 BSP 模型快 22%、21%、39%、30%、28%；比 AP 模型在 64-fragment 快慢 0.78%在 96-fragmnet、128-fragment、160-fragment、192-fragment 下快 5%、27%、23%、29%；比 AAP 模型在 64-fragment、96-fragmnet、160-fragment 下慢 8%、6.5%、0.24%，在 128-fragment 、192-fragment 下快 9.9、15%；可以看到 MPAP 模型比 BSP 于 AP 模型具有较大的性能优势，与 AAP 原型相比，在分区数为 64，96 下性能不理想，但随着分区数的增加，性能超过 AAP，快 15%。

可以看到在 bfs、hits、pagerank、louvain、sssp、wcc、wcc_hashmin 等算法 MPAP 较其余计算模型具有不同程度的性能优势，但在与 lpa 算法中，BSP 的性能优势更加明显。

3 下一阶段工作计划

3.1 论文研究进度

选取很多工业界与学术界常用的算法，并针对算法在 PIE 模型下的运行时特点进行分类	是

针对每一类算法，给出不同的特征提取方案	是
针对每一类算法，测试多种回归模型，以比较不同模型在不同算法上的优缺点	是
开发 MPAP 模型系统	是
针对上述算法，比较 MPAP 模型与 APBSP AAP 模型的性能	目前已经完成大部分算法比较，MPAP 在慢机情况下，大部分算法都有更好的性能优势，但有有几个算法优势不大，还在进一步分析原因

3.2 尚未完成工作

1. 模型的训练上还差神经网络模型没有给出最终实验结果，因为神经网络模型较复杂，涉及参数较多，需进一步找到最佳的参数组合，之后给出结果。

2. 没有一个用于展示的 web 页面服务，且所做工作涉及的内容分散零散，需最终整合，用于更好的展示.

3. 一些算法在 MPAP 上性能较 AAP 有一些劣势，后续会深入查找原因。

3.3 下一阶段计划

针对已有的实验数据，完善下数据可视化展示；针对没有完成的实验数据，尽快补足数据并进行展示，并进行最终论文的撰写。

4 主要参考文献

[1] https://en.wikipedia.org/wiki/Connected_component_(graph_theory)

[2] Frigioni D, Marchetti-Spaccamela A, Nanni U. Fully dynamic output bounded single source shortest path problem[C]// Acm-Siam Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 1996:212-221.

[3] Brin S, Page L. The anatomy of a large-scale hypertextual Web search engine[J]. Computer Networks, 2012, 56(18):3825-3833.

[4] Google. How search works. http://www.google.com/insidesearch/howsearchworks/thetory/

[5] Pržulj N. Protein-protein interactions: making sense of networks via graph-theoretic modeling.[J]. Bioessays News & Reviews in Molecular Cellular & Developmental Biology, 2011, 33(2):115-123.

[6] Siek J G, Lee L Q, Lumsdaine A. The boost graph library: user guide and reference manual[M]. Addison-Wesley Longman Publishing Co. Inc. 2002.

[7] Gregor D, Lumsdaine A. Lifting sequential graph algorithms for distributed-memory parallel computation[C]// Acm Sigplan Conference on Object-oriented Programming. ACM, 2005:423 437.

[8] Dean J , Ghemawat S . MapReduce: simplified data processing on large clusters[M]. ACM, 2008.

[9] Apache Software Foundation. Apache Hadoop [EB/OL]. https://hadoop.apache.org/ , 2013

[10] Apache Giraph. http://giraph.apache.org/

[11] Salihoglu S , Widom J . GPS: a graph processing system[J]. 2013.

[12] Spielman D A, Teng S H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems[C]// 2004:81-90.

[13] Bickson D. GraphLab: Asynchronous Graph Computation in the Clouds and Beyond[J]. Archives of Biochemistry & Biophysics, 2006, 452(2):138-148.

[14] Xie C, Chen R, Guan H, et al. SYNC or ASYNC: time to fuse for distributed graph-parallel computation[C]// Acm Sigplan Symposium on Principles & Practice of Parallel Programming. ACM, 2015:194-204.

[15] Fan, Wenfei, Xu,LuPing, XiaoJian, Luo, et al. Adaptive Asynchronous Parallelization of Graph Algorithms[J]. Proceedings of the SigMod, 2018, 6(10): 1141-1156