Cost-Effective App Data Distribution in Edge Computing

Xiaoyu Xia , Feifei Chen , Qiang He , Member, IEEE, John C. Grundy , Senior Member, IEEE, Mohamed Abdelrazek, and Hai Jin , Fellow, IEEE

摘要

边缘计算作为云计算的重要补充，其核心思想是任务前移，通过计算和存储资源从集中式云分配到分布式边缘服务器，使得数据能在接近应用端的节点上完成计算来支持各种需要低延迟的应用. 例如，物联网服务、虚拟现实、实时导航等. 从应用供应商的角度来看，云边协作模式下，应用程序需要将云中的数据传输到某个区域指定的边缘服务器来服务该区域的程序用户. 而按需付费的商业模式下，将大量数据从云分发到边缘服务器可能很昂贵. 复现原文致力于研究满足延迟约束下成本更低的边缘数据分发策略. 云边协作场景下，数据分发成本包含云到边缘服务器之间和边缘服务器之间的成本，同时必须满足延迟约束-数据分发不能花费太长的时间. 据复现原文描述和我查阅的资料证实，这确实是第一个尝试将边缘数据分布 (EDD) 问题构建为约束优化问题，证明其是 NP-Hard 问题并提出有效分发策略的工作. 原文提出 EDD-IP 策略利用整数规划思想求解最优分发策略，针对大规模数据分发场景，提出 EDD-A 算法近似求解有效分发策略. 此次工作复现对 EDD-IP 和 EDD-A 在现实数据集 EUA 上的实验评估，验证了原文中的算法确实显著优于随机策略、贪心策略、最低成本多播路由策略 (MMR) 这三种经典数据算法的结论，并补充在随机图上的测试并分析 EDD-A 算法的劣势和探讨可行研究方向.

关键词：边缘计算, 最优化, 数据分发, 成本效益分析, 边缘服务器网络

1 引言

在这十年中，移动设备和物联网 (IoT) 连接设备的大量增加推动了移动数据流量的成倍增长. 移动数据流量的激增导致了大量研究，旨在减轻网络上巨大的数据传输负载. 由云计算推动的传统网络范式，包括内容交付网络、以内容为中心的网络和以信息为中心的网，无法处理由快速增长的移动流量导致的网络延迟和网络拥塞的增加. 近年来，边缘计算已成为一种新的计算范式，将计算和存储资源推到云的边缘[1]. 这些边缘服务器由一台或多台物理机器供电，部署在地理位置接近设备的基站或接入点[2]. 随着边缘服务器成为更多移动和物联网设备进入互联网的入口，越来越多的快速增长的移动流量数据将通过这些边缘服务器从云端传输. 此外，这将大大减少云与应用用户之间传输的应用数据量. 从应用程序供应商的角度来看，在边缘服务器上缓存应用程序数据可以大大减少其用户数据检索的延迟. 这反过来将降低相应的数据传输成本. 边缘计算环境中数据缓存带来的新挑战吸引了许多研究人员的注意[3-7]. 然而，现有的研究工作集中在如何跨边缘服务器缓存数据以实现不同的优化目标，例如，最小化缓存成本[6]，最小化检索延迟[7]，保证传输质量[4]等. 忽略了云数据分发的策略也是可以降低成本的重要策略. 从云服务器应用供应商的角度来看，数据激增导致数据传输代价增大，特别是云数据大量分发到边缘服务器时成本十分昂贵. 例如，亚马逊供应商将数据从 S3 云数据库传输到互联网的费用是:0.09- 0.11$/1GB，而传输的数据往往十分庞大[1]，自动驾驶汽车 1.5 小时就可以产生 4TB 数

[1]https://cloud.tencent.com/developer/article/1145206

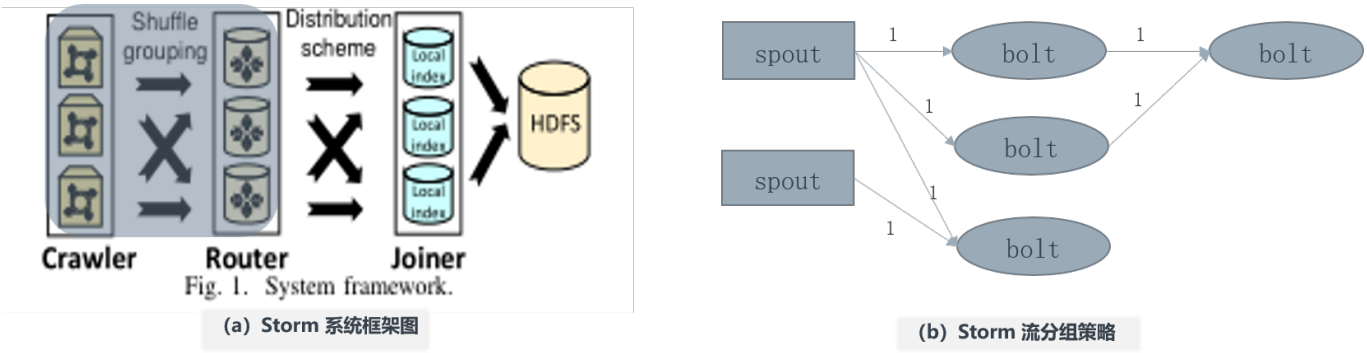


图 1: Storm 平台相关处理概念图

据[2]. 因此，研究数据分发问题具有很现实的商业价值. 文章复现的工作主要包括:

- 原文首次尝试从应用程序供应商的角度将边缘数据分布 (EDD) 问题建模为约束优化问题，并证明是 NP-hard 问题.
- 利用 python 语言实现原文提出求最优解的 EDD-IP 回溯算法和求近似解的 EDD-A 的 $O(K)$-优化算法，并给出伪代码.
- 追溯对比算法原文并分别实现基于随机策略、贪心策略以及多路组播场景下的 MMR 近似算法.
- 依照原文要求在 EUA 真实数据集[3]上构造实验数据，观察五种算法的准确性和运行时间. 此外，复现时对文章算法在随机图上进一步验证有效性.

近期我在学习 Storm 平台的流数据处理 (如 1(a)). 初衷是我想详细了解第一阶段数据流分组（即数据分发策略）过程，将这篇文章做拓展学习.Storm 的第一阶段 Shuffle grouping : 把 Crawler(相当于 spout) 抓取到数据平均分组分别发送到指定 Router(相当于 bolt). 但 EDD 问题与 Storm 数据流分组问题仍存在异同点, 相同点是 Shuffle grouping 分发与本文的核心问题性质一致. 每个组的数据量基本一致，相当于把一个数据组重复分发到多个 Router(如 1(b)). 不同在于 Storm 平台中传输代价是一致的. 文章的 EDD 问题存在 E2E 和 C2E 不同的代价问题，而 Storm 平台中 spout-bolt 或 bolt-bolt 的传输代价是一致的. 我进一步的考虑发现本文的 EDD-IP 和 EDD-A 问题对图节点有特定的要求. 图中节点度比较大的节点不宜集中在 $\mathcal{R}$ 集合中. 在复现后我对随即图进行测试证明了我的猜测. 当图中度比较高节点几乎在 R 集合中时，EDD-A 不具备优越性，因为根据节点度选取分发节点的贪心策略可以很快速拿到最优解.

2 相关工作

2.1 云边协作模式

云边协作工作模式核心是将计算资源从集中式云分发到分布式边缘服务器，如图 2(a)所示加一层边缘云，可以大大降低应用程序延迟，数据再集中汇总到中心云进行数据分析等工作以调整边缘服务器的策略或状态，更好地服务于实时导航、虚拟现实、物联网应用等. 例如：Facebook Horizon 的 VR games 和 VR vedios 应用在云边协作模式下才有可行性. 云边协调模式导致数据流量传输成本激增，从图 2(a)看出中心云和边缘云需要数据双向传输. 边缘 -> 云：各个边缘服务器处理后数据不同，需要通

[2]https://zhuanlan.zhihu.com/p/137748742

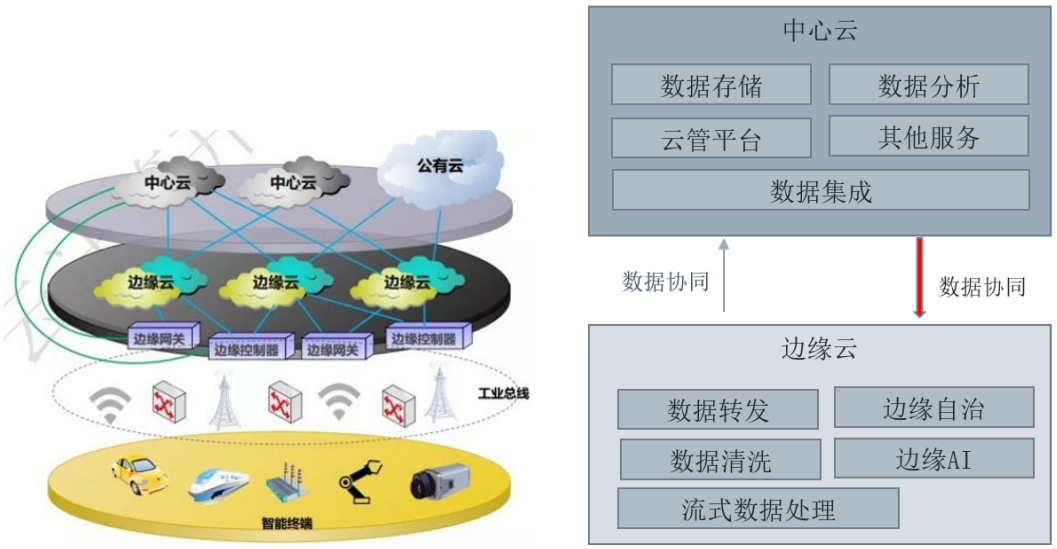
[3]https://github.com/swinedge/eua-dataset

过光纤上传到中心云集中存储和分析. 云-> 边缘：因为跨区数据交互或账号同步等需要中心云将同一数据分发到多个指定的边缘服务器中. 云层数据分发到边缘层会产生昂贵的费用. 如何将云数据分发到一个边缘云内多个指定的边缘服务器，这正是原文所解决的难题.

2.2 边缘计算相关研究

边缘云只有在中心云配合的时候才有其作用，因此边缘计算的研究总是和中心云相关的，我们把云边协作场景下的研究问题归为边缘问题. 边缘计算的研究方向包括：边缘服务器放置、边缘用户分配、服务器卸载、边缘应用程序部署、边缘数据缓存、边缘数据分发.

针对于边缘服务器放置问题，Yao 等人[8]提出基于整数编程的边缘服务器放置策略. Yin 等人[9]提出 Tentacle 策略，最大限度降低部署成本同时最大程度提高性能. 针对于边缘用户分配问题，Lai 等人[10]建模为可变大小的装箱问题，最大化分配用户数量同时降低租赁服务器成本.he 等人[2]提出 EU-AGame，利用分散算法找到纳什平衡作为近似解. 针对于服务器卸载问题，xu 等人[11]提出在线算法 OREO，利用服务器缓存降低卸载延迟同时尽量保持低能耗.Wang 等人[12]将问题建模为凸包问题，分治法解决分布式问题. 针对于边缘应用程序部署问题，He 等人[13]提出基于最大化用户或最大请求的最佳部署策略.Wang 等人[14] 考虑在云内应用迁移代价，提出基于最小化部署成本的部署策略，目的是最小化平均迁移成本和传输成本. Mahmud 等人[15]基于新的定价模型和用户补偿方法，提出了一种找到满足资源约束（如处理核心和内存）的最优应用程序部署策略. 针对于边缘数据缓存问题，Drolia 等人[3]提出 Cachier 缓存系统，用于协调边缘服务器和云之间的负载平衡，以动态方式最小化数据检索延迟.Breitbach 等人[16]提出基于任务调度的边缘解耦数据管理系统.Shao 等人[17]将问题建模为 0-1 整数规划模型，提出一种智能群优化变体的算法. 针对于边缘数据分发问题，文章指出目前没有专门针对边缘数据分发 (EDD) 问题的研究.EDD 问题的特点: 云到边缘的代价一致，而边缘之间的传输代价一致. 这不同于每段传输代价都不一致的路由分发，因此探索更高效的分发算法.



(a) 云边协作模式网络框架图　　(b) 云边协作分工模式

图 2: 云边协作

3 本文方法

3.1 EDD 问题描述

在边缘计算环境中，部署在不同基站和接入点的相邻边缘服务器可以与其相邻边缘服务器通信，并通过高速链路共享其存储资源. 原文以 Facebook Horizon 的多玩家 VR 协作场景为例，VR 用户拥有独立账号且分散各地，由多个附近的边缘处理器和一个共享云端完成协同处理，如图 3. 按需付费模式下，数据传输价格大约为 0.1$/1GB，对于大型应用来说数据传输代价十分昂贵. 这正是文章研究的实际问题. 以亚马逊为例，供应商将 1 GB 数据从 S3 数据存储库[4]传输到互联网的费用高达 0.09 美元 +0.02 美元. 而传输的数据往往十分庞大，自动驾驶汽车 1.5 小时就可以产生 4TB 数据.[5]. 针对 Horizon 场景文章给出了一个更具体的场景，假设 Facebook Horizon 云上一个 VR 视频需被缓存到其中的 7 个指定边缘服务器上, 如图 4左. 文章问题进一步具体为：如何针对 Horizon 云分发问题定制成本最低的分发策略？（我查阅资料发现，Facebook VR 协作平台 Horizon Workrooms[6]原本预计在 2020 年发布，但是最终在 2021 年发布成果，而这篇文章在 2021 年 3 月被 TPDS 接收的，猜测可能原因之一是原文解决的传输代价过高问题.)

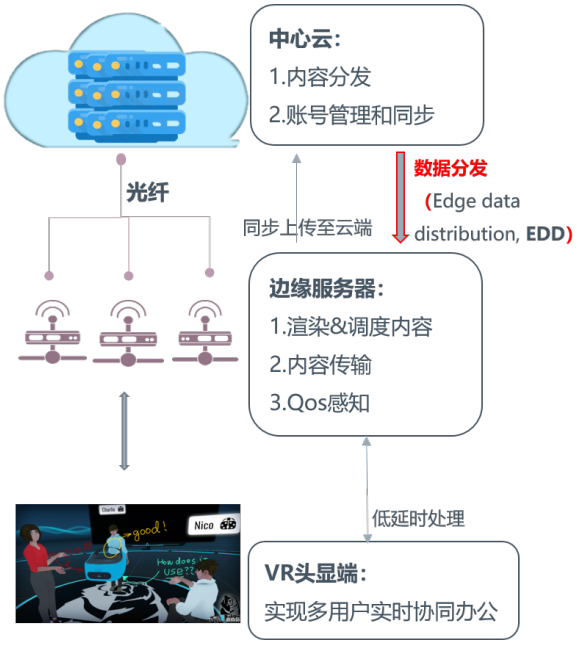


图 3: Facebook Horizon 云边协同模式

文章对 Facebook Horizon 上一个 VR 视频的缓存问题建模 (EDD). 文章将传输类型分为两种：云到边缘，即 C2E 传输；边缘服务器之间的传输，即 E2E. 一般情况下，同等数据量的情况下 C2E 的代价远高于 E2E.VR 应用最核心的就是实时性，EDD 须遵循由应用供应商提供的时间约束，该 EDD 问题的时间延迟由 E2E 的跳数衡量. 文章对 EDD 问题建模:

1. 设同等数据量传输下，$Cost(C2E) = \gamma * Cost(E2E), \gamma > 1.$

2. 设 S 表示直接收到云端数据的初始节点集合，$\mathcal{R}$ 表示所有指定缓存的边缘服务器集合.d_v 表示节点 v 的时间延迟，d_limit 表示允许的最大延迟.

建模后可以将问题转化为约束寻优的图计算模型，如图 4右所示. 本文问题的目的是为了降低将云数

[4]https://aws.amazon.com/cn/s3/

[5]https://www.infoobs.com/article/20200818/41392.html

[6]https://zhuanlan.zhihu.com/p/401744641

据分发到一个边缘云内多个指定的边缘服务器的代价，因此目标方程为：

$$minimize\ \ Cost_{C2E}(S_{C2E}) + Cost_{E2E}(\Gamma_{E2E}) \tag{1}$$

同时满足 $\forall v \in \mathcal{R}, 0 \leqslant d_v \leqslant d_{limit}$

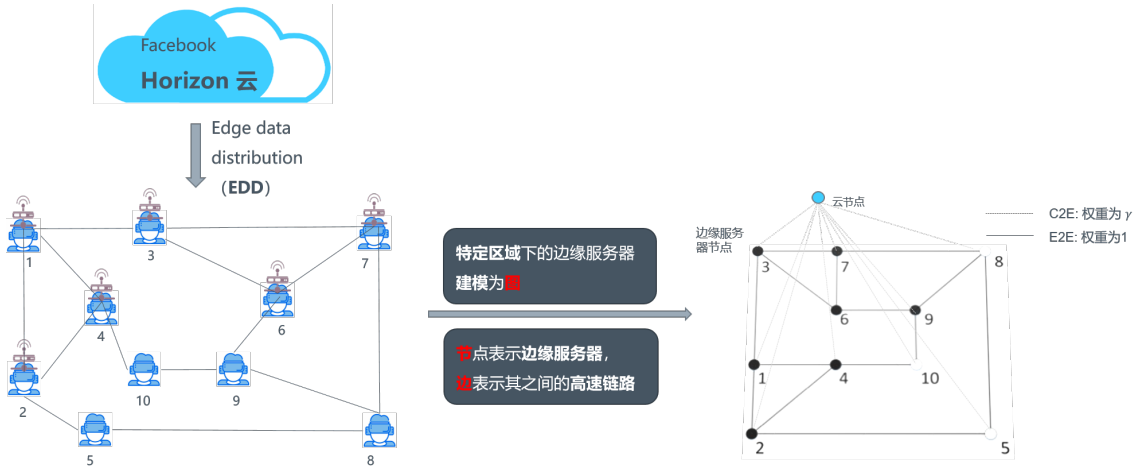


图 4: Horizon 云分发的一个 VR 视频的例子

NP-Hard 证明 文章提出 EDD 问题可以归约到指定根节点的最小斯坦纳树 (Rooted Minimum Steiner Tree , RMST) 问题，证明 EDD 是 NP-Hard 的. 也就是说，EDD 问题本质上是加了延迟条件限制的 RMST 问题 (NP-Hard 问题)，即考虑如何将云节点作为根节点使得数据到多个指定的边缘服务器（不一定连通）代价最小.（这一部分感兴趣的可以查看原文）

3.2 本文方法概述

此部分对本文将要复现的工作进行概述，文章首次发掘边缘数据分发问题并证明了其不可能在多项式时间内得到最优解. 因此文章提出优化方法 EDD-IP: 给出约束范式，通过调用整数规划工具 CPLEX 的机器学习方法逼近得到最优解. 这一模块调用 pyhton 包（cplex 库）后可以得到最优解. 实在没有技术含量，因此复现时，我尝试基于 EDD-IP 提出的思路利用回溯法得到最优解.

针对大规模网络图，文章提出 EDD-A 近似求解算法解决数据传输问题, 通过快速找到一个方法并调整为近似最优策略. 该方法启发于 Minimum-cost Multi-cast Rousting(MMR) 路由算法. 文章并证明是 EDD-A 是 K 近似算法，即近似误差是常常数级的.

最后，通过和经典的路由算法 Random、Greedy、MMR 路由策略进行对比试验验证 EDD-A 有效性.

4 复现细节

复现工作 该文章与 21 年在 TPSD 会议上发布并没有开源代码，网上也很少查询到关于此文章的相关帖子等信息，因此未知其所用编程语言. 我在实现中选择 pyhton 实现，很大一部分原因是因为 python 上图的可视化方便，有利于调试代码.

复现内容上, 我利用 python 实现文中提出的 EDD-IP 精确算法，EDD-A 近似算法；以及实现对比的 Random、Greedy、MMR 分发算法. 利用文章用到的 EUA[7]数据库实现小样本上包括 EDD-IP 的 5 种算法在时间、准确性上的消融实现.

[7]https://github.com/swinedge/eua-dataset

复现难点上，首先是文章中 EDD-IP 未给出伪代码. 复现时我只好根论文思路，尝试回溯找到最优解. 其次，实验数据很难搞，关于文章的具体的实验数据基于 EUA 数据集的构造没有详细介绍，仅仅是提到通过 ER 随机模型对数据集进行采样，我无法了解到文章 ER 随机采样的具体参数以及如何将 EUA 数据集提供数据处理为可行的输入（发邮件问询原作者还未回信）. 最后我只好通过其他使用 EUA 数据集文章借鉴其进行的实验进行补充.

补充实验 我进一步针对小规模随机图 (非 EUA 数据集) 进行 5 中算法测试. 观察 EDD-A 和 EDD-IP 的有效性正确性和时间, 并针对 EDD-A 策略表现较差的情况进行分析. 实现代码框架如图 5所示：

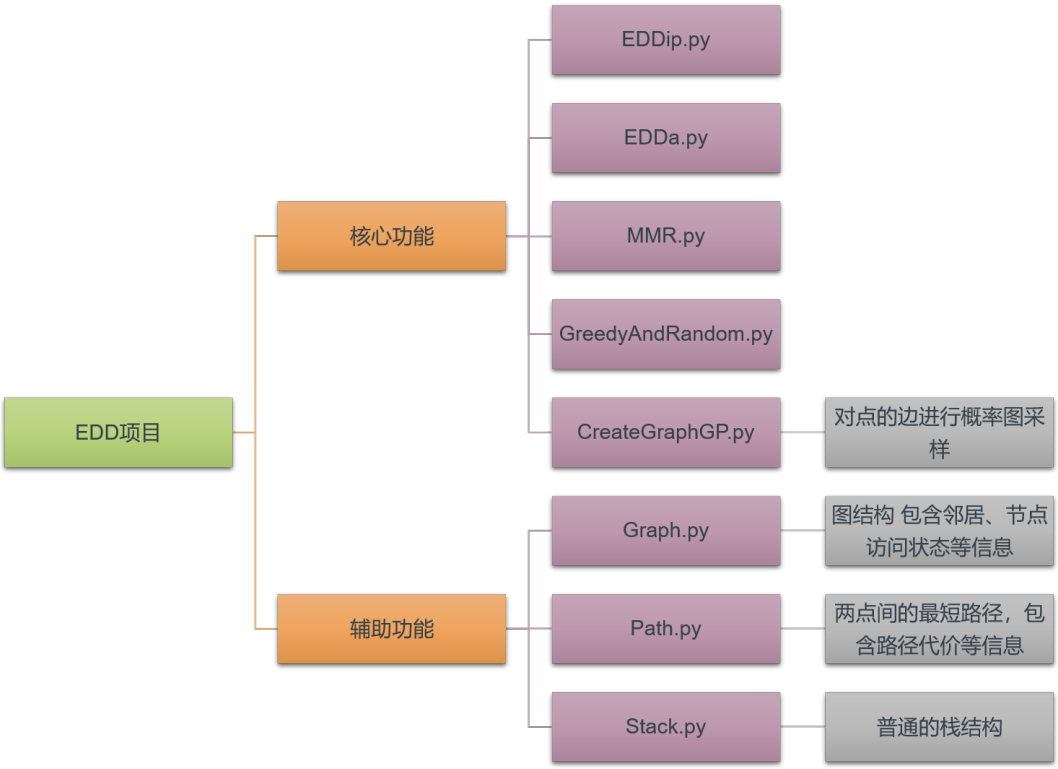


图 5: EDD 代码框架图

4.1 EDD-IP 算法实现细节

这部分根据论文思路，在复现时尝试回溯法求解. 针对 EDD 问题算法的输入：图连接关系 G，指定接收边缘节点 $\mathcal{R}s$ 集合，最大延迟限制 d_{limit} ,$C2E$ 的传输代价 γ.EDD-IP 输出：找到全局最优分发策略/未找到可行策略.EDD-IP 算法大致分为两步：

- ✏ 找到直接和云节点相连的节点组成 S_{C2E} 集合, 集合中的边缘服务器接受云的传输，代价为 γ
- ✏ 然后从 S_{C2E} 集合中的各个节点**扩散**直至访问到所有的 $\mathcal{R}$ 节点或者违反最大延迟限制则结束. 扩散实现细节如算法 1所示.

Procedure 1 扩散函数 Fillout.

Input: 图的邻接矩阵 G, 当前策略 p，节点访问状态 $visit$，当前树高 d，策略代价 $cost$
Output: 更新当前策略 $path$ 和代价 $cost$

```
if d ⩾ d_limit then
    退出，返回 cost                    ▷ C2E 节点的 d = 0□d_limit = 2 表示边缘节点之间只能传输 1 次.
end
for v in G.vertexs do
    for u in v.adjcents and v.isVisit=True    ▷ 对被访问过节点 v 的所有邻居节点进行遍历 do
        if u.isvisit=False and u.isR=True then
            u.isvisit=True  p.append(u)  cost += weight_uv)    ▷ 更新策略和代价
        end
    end
end
allVisited=True                                ▷ 判断是否访问完所有的 r 节点
if v.isR= True and v.isVisited=False then
    allVisited=True  break
end
if allVisited=False then
    Fillout(G,p,d+1,cost)                      ▷ 没访问完就继续广度扩散
end
```

回溯的核心就是遍历所有的可行的 S_{C2E} 集合组合找到最小成本的分发策略. 实现过程中图采用邻接表存储，因此 EDD-IP 的算法时间复杂度是 $O(2^V * (V + E))$, 其中 V 是节点个数，2^V 是 S_{C2E} 集合种类数. EDD-IP.py 的实现细节如图 6：

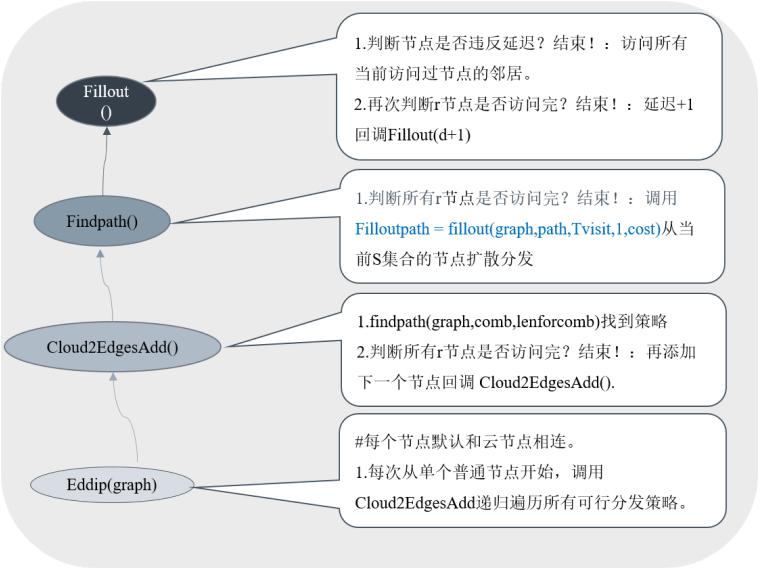


图 6: eddip.py 实现核心逻辑

以 Horizon 云 4的问题为例，EDD-IP 的运行过程如图 7：Eddip 回溯的所有 S 集合组合，每一个 S 集合都可以展开成一棵回溯树. 当 S=1,6 时找到最优的一个解 $cost = 2\gamma + 5$. **时间复杂度上**, 因为节点个数为 $v = 10$，所以 S 集合个数 $=C_{10}^1 + C_{10}^2 + .. + C_{10}^{10} = 2^{10} - 1$, 对于每个集合 S，需要遍历所有的节点 $T(V) = O(V + E)$，因此 EDD-IP 总的时间复杂度为 $T(V) = O(2^V * (V + E))$

4.2 EDD-A 算法实现细节

图的存储结构为邻接表，因此 EDDA 时间复杂度：$O(V^3 + E)$,V 是节点个数,E 是边数. 这部分按照原论文伪代码实现.EDDA 找到的是近似最优的分发策略.EDDA 核心思想是先通过 CMST 算法 2找到一个可行的 ST 树，然后通过调整 ST 树中每个不满足延迟约束的节点，得到可行近似最优解.Edda.py

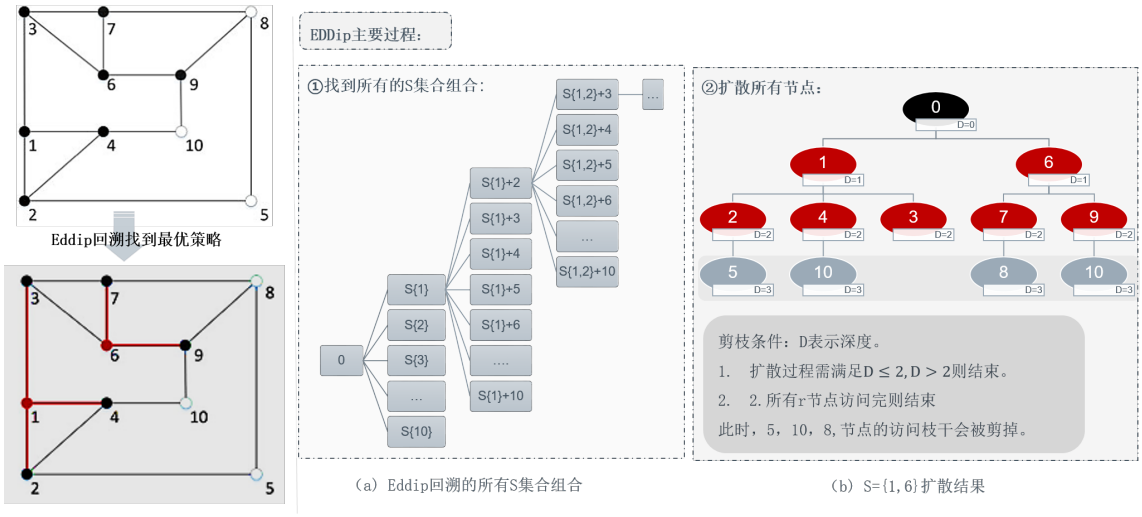


图 7: eddip 求解 horizon 分发问题案例

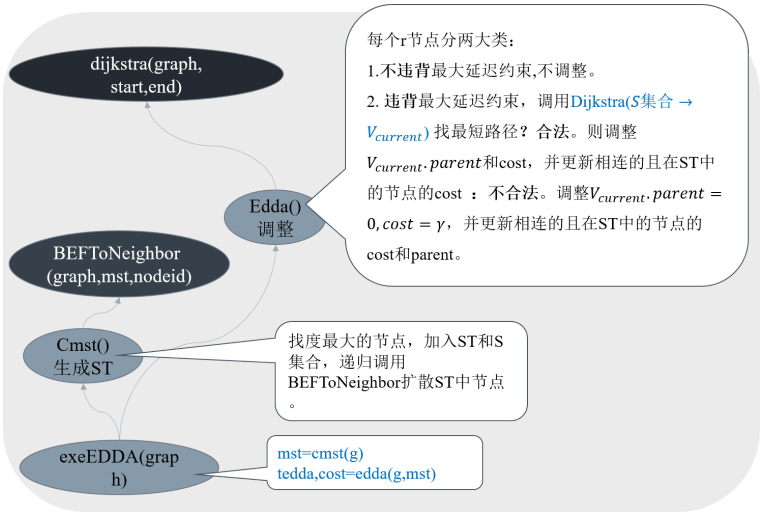


图 8: Edda.py 实现核心逻辑

功能首先主要逻辑如图 8.

✏ CMST 算法 (如 Procedure 2)：每次都找整个图中度最大的节点加入到 S 集合和 ST 树中，然后扩散 ST 中节点，将其邻居节点加入（ST 树）. 重复操作，直至所有的 r 节点都包含在 ST 树中.

✏ EDD-A 调整 (如 Procedure 3)：依次遍历 ST 树中所有节点，如果不违背 d_{limit} 则不调整，如果是 r 节点且违背延迟约束，则通过 Dijsktra 算法找到 S 集合中节点到该节点的最短路径，如果不违背 d_{limit} 则更新该节点的分发前驱节点，否则将该节点调整为 C2E 边节点，加入到 S 集合中.

Procedure 2 建立最小子树 CMST

Input: 图的邻接矩阵 G
Output: 最小斯坦纳树 tms
初始化 tms=cloud ▷ 此时 tms 只有一个云节点，云节点和所有节点有连边 tmsallR=Fasle ▷ 判断是否所有 r 节点都在 tms 中
if *tmsallR=True* **then**
  return tms
end
while *tmsallR=Fasle* **do**
  v $\leftarrow G.findmaxConnect\ and\ v.isR = True$ ▷ 找到 G 中的最大度的 r 节点
  v.isvisite $\leftarrow True$
  $tcost \leftarrow BEFToNeightbor(G, tms, v, limit = False)$ ▷ 最大度 v 节点尽可能扩散到所有相邻的 r 节点，包括邻居邻居的 r 节点. 无设置 d_{limit} 限制 (Procedure 4)
  for V *in* $G.Rlist$ **do**
    if *v not in tms* **then**
      tmsallR $\leftarrow False$
      break
    end
  end
end
return tms

Procedure 3 调整 EDDa

Input: 图 G，近似最小生成树 mst，延时阈值 d_{limit}, C2E 代价 γ
Output: 调整后满足延时约束的 tms 树
初始化云节点编号为 0，遍历 mst 中每个节点 v_cur
if $v_{cur} isR = True v_{cur} d \leqslant d_{limit}$ **then**
  continue
end
if $v_{cur} isR = True v_{cur} d \geqslant d_{limit}$ **then**
  遍历 S 集合中每个节点，计算到 v_cur 的最短路径 $S, path = Dijsktra(s_i, v_cur)$
  if $len(path) \leqslant d_{limit}$ ▷ 说明找到合法路径 **then**
    根据 $path$ 更新 $v_{cur} parent, cost, depth.$
    更新 $\forall u \in tms\ and\ connetc(v_cur, u) = True, u_{cost}, u_{depth}$
  else
    v_{cur}.parent=0 ▷ 将当前节点直连云节点
    更新 $\forall u \in tms\ and\ connexct(v_cur, u) = True, u_{cost}, u_{depth}$
  end
end

以 3.1 章节提出 horizon 云场景问题为例，CMST 和 EDDA 调整过程如图 9所示，调整过程是文章的关键：对于违反了 dlimit 的节点，我们首先查找一个 cloud-other-v 路径，如果路径找到一个可行的让 V 不违背 dlimit 的父节点，那就更新 v 的父节点和 cost，depth，并更新 v 的相连的且在 CMST 中的节点的 cost，depth 信息. 如果找到的路径上 V 依旧违背 dlimit，那么把 V 设置为 cloud-V 的节点，直接和 cloud 相连. 然后更新 graph 上和 V 相连的所有节点信息 parent cost depth 信息（添加一个行的 source 节点后，可能是旧 source 子树尾部的 r 节点会被并入到新 source 根节点的树中）.CMST 得到的 tms 树的连接关系如图 9(a) 所示，EDD-A 调整过程如图 9(b) 所示，最后的策略中，节点 1，6 直接连接云节点，1 分别连接 2,4,3 节点，6 扩散到 7,9 节点. 总的传输代价为 cost=$2\gamma + 5$.

EDD-A 算法复杂度上，节点个数为 v=10, 由于是图是邻接表存储的，CMST 的主要操作就是找最

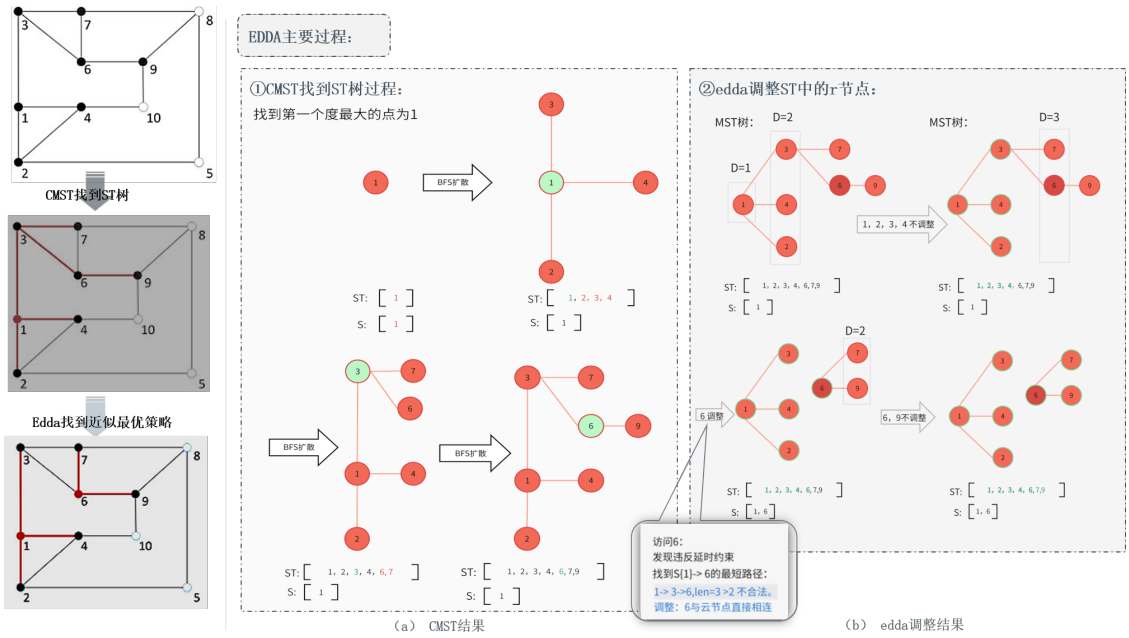


图 9: edda 近似求解 horizon 分发问题案例

大度节点 $O(V)$ 以及 BFS 扩散操作，因此 R 个 r 节点，最坏情况下需要查找 R 次，而每一次零散地扩散处理完所有节点后相当于一次完整的 BFS 遍历，因此 CMST 整个过程的复杂度是 $O(VR+(V+E)) = O(V^2+E)$;Edda 调整按序号大小遍历 ST 树，违反需要 Dijkstra 找最短路径 $O(V^2)$ 并更新其所有属于 tms 节点的邻居节点, 因此 V 个节点的调整复杂度为 $O(V^3)$. 所以,EDD-A 算法总的时间复杂度为 $O(V^3+V^2+E) = O(V^3+E)$.

4.3 对比算法实现细节

这部分按照原论文提及思路实现. 随机算法的核心思想是：每一次随机选取一个边缘节点，C2E 传输后传输相邻的 r 节点，在不超过最大延迟条件下 BFS 扩散. 重复操作，直到访问所有的 r 节点. 其核心代码如 Procedure 4所示. 随机选取次数最多 V 次，复杂度为 O(V), 每选取 1 次就要进行扩散，因为有延迟约束，每次扩散会被剪枝, 总的扩散过程相当于 1 次完整的 BFS 遍历 $O(V+E)$. 因此随机总的复杂度为 $O(V+E)$.

Procedure 4 Random

Input: 图 G，延迟阈值 d_{limit} , 指定节点序列集合 $\mathcal{R}$，节点个数 V
Output: 策略代价 cost

初始化 isfinish $\leftarrow False, cost = 0$ ▷ 程序结束标识
while *isfinish* = *False* **do**
  for v *in* $\mathcal{R}$ **do**
    if *v.isviste* = *False* **then**
      isfinish $\leftarrow False$ ▷ 如果访问完所有的 r 节点就退出程序
    end
  end
  id $\leftarrow random.randint[1\ V]$ *and* $G(id).isvisite = False$ ▷ 随机选取一个节点
  G(id).isviste $\leftarrow True$
  $cost+ = \gamma + BDFToNeighbor(G, id, 1, limit = True, d = 1)$
end
return cost

贪心分发策略的核心思想是：每次选取度最大的未访问节点，C2E 传输，然后在延迟约束下最大限度扩散到邻接的 r 节点. 重复操作，直到访问所有的 r 节点. 当指定节点个数等于总节点数，即

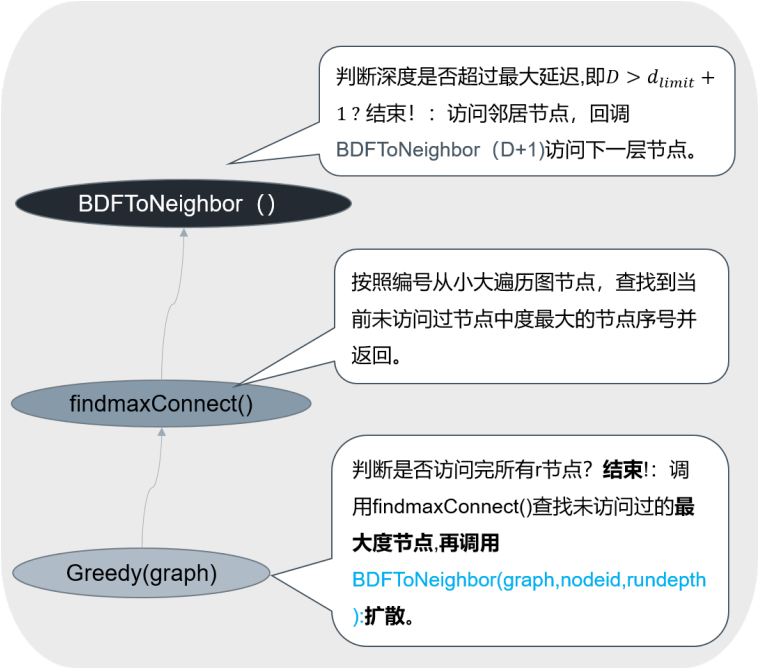


图 10: Greedy 主要实现逻辑

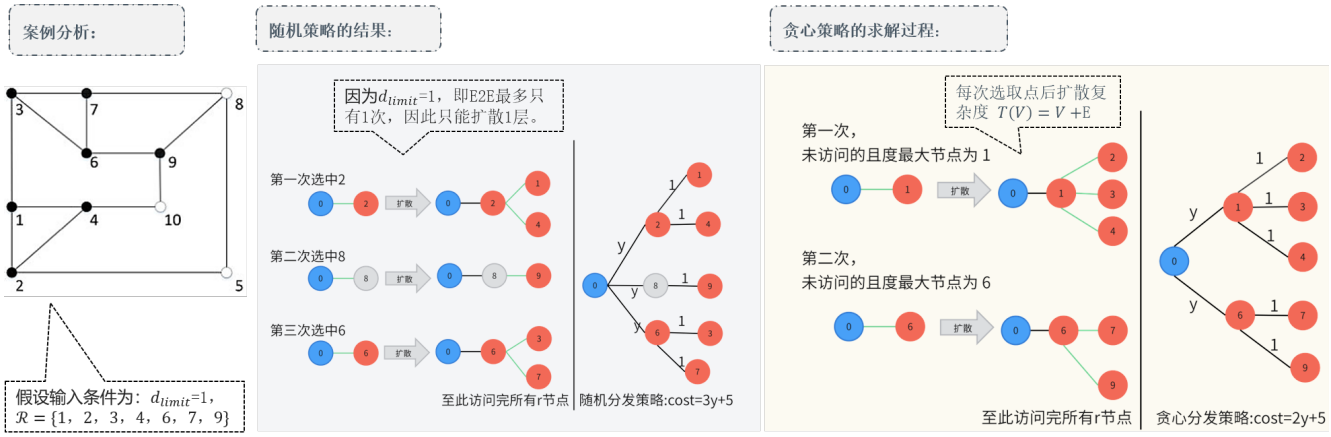


图 11: random 和 greedy 求解 horizon 分发问题案例

$|\mathcal{R}| = V$ 时，最坏的情况下需要选取 V 次. 因为有约束限制，总的扩散相当于一次完整的 BFS 遍历. 因此贪心总的复杂度是 $O(VR + V + E) = O(V^2)$.Greddy.py 的核心实现逻辑如图 10.

以 3.1 章节提出 horizon 云场景问题为例，最大延迟 $d_{limit} = 2$. 随机和贪心策略的执行过程如图 11所示. 其中，假设随机过程选择的节点分别为 2，8，6，BFS 扩散后将 VR 视频成功传输到所有指定的边缘服务器，总的随机代价为 $cost = 3\gamma + 5$. 而贪心策略优先选择度最大的 r 节点，分别选择到度为 3 的 1 号节点，扩散到 2，3，4 节点后达到最大延迟，因此需要再新的未访问过的度最大的节点，第二次选择到度未 3 的 6 号节点，扩散到 7,9. 至此发现已经达到最大延迟，且已经访问完所有的 r 节点，返回总代价 $cost = 2\gamma + 5$.

MMR 这部分按照原文引文 [28] 实现的.MMR 分发和 EDDA 相似，分两阶段: 第一阶段也是 cmst 找到最小的斯坦纳树，第二阶段，对于不合法的 r 节点，MMR 找到从源节点-> 当前节点的最短路径. 可见 MMR 算法在路由转发场景下是可行的. 但在 EDD 场景下，MMR 中违反的节点将直接和云节点相连，显然不是最优的.MMR 的算法复杂度也是 $O(V^3)$. 图用邻接表存储的. 第一阶段: 顺序查找顶点数组得到最大度节点序号 $O(V)$，最多 R 次. 整个扩散过程相当于一次完整的 BFS 遍历 $O(V + E)$, 因此该阶段复杂度为 $O(VR + V + E) = O(V^2 + E)$, 第二阶段：遍历每个节点，对每个不合法节点，每

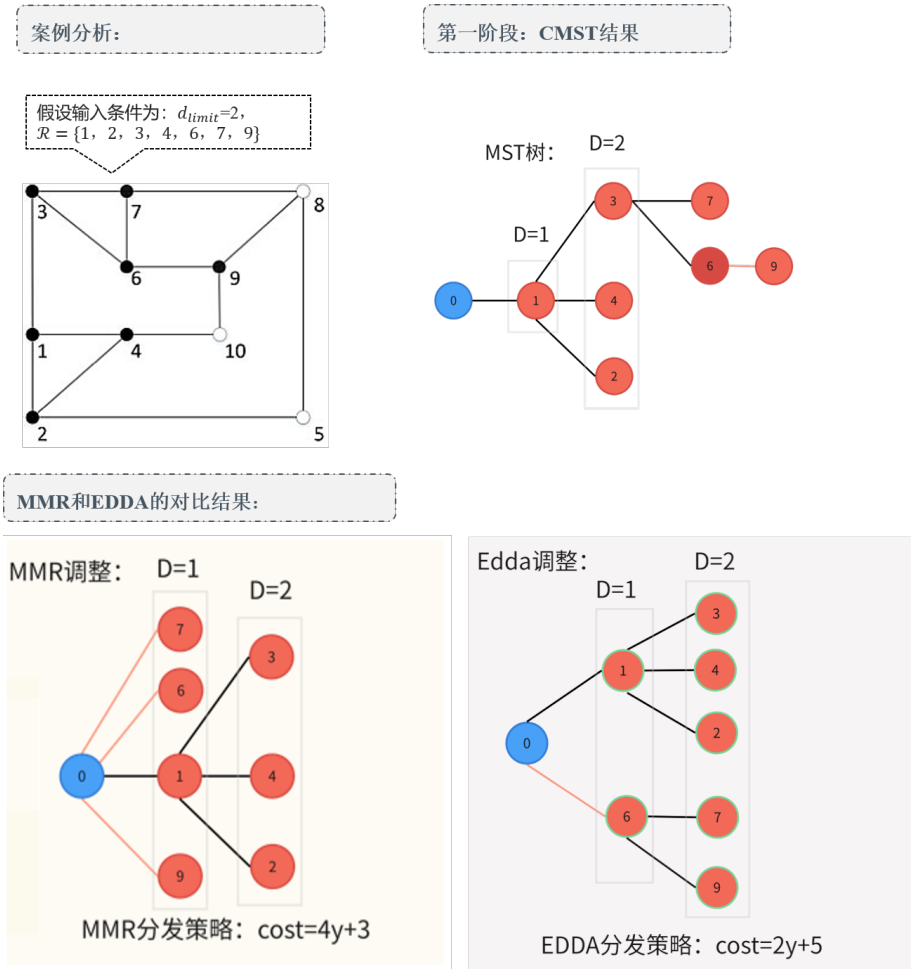


图 14: MMR 和 EDDA 基于 horizon 分发问题的对比结果

次都要调用 Dijkstra 算法 $O(V^2)$, 最多 R 次，因此 MMR 总的调整时间复杂度为 $O(V^2 * R) = O(V^3)$.

因为 MMR 和 EDDA 相似，针对 3.1 提出的 horizon 现实分发场景，对 EDD 和 MMR 的结果作对比，可以发现 MMR 不适合边缘分发场景，MMR 没有充分利用图权重只分为 C2E 和 C2E 的条件. 对比结果如图 14所示，MMR 调整过程只是将不合法的节点直接连接到云节点，最后总的传输代价十分高 $cost = 4\gamma + 3$, 现实场景中 γ 一般大得多，因此 MMR 的代价远超于 EDDA 得到的 $cost = 2\gamma + 5$ 分发策略.

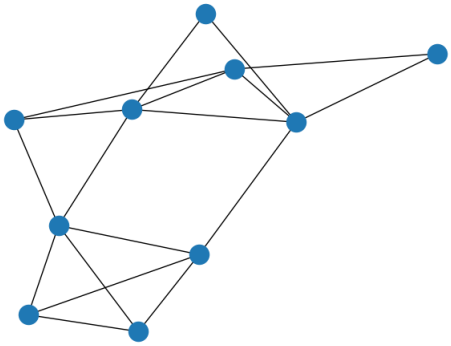


图 12: 随即图采样得小规模图可视化

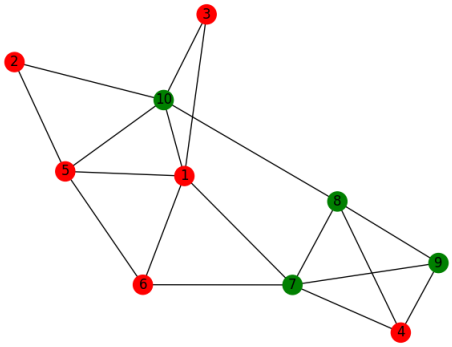


图 13: edd 问题的图可视化

Procedure 5 BEFToNeighbor

Input: 图 graph, 斯坦纳树 tms, 当前扩散节点 nodeid, 选填传入 limit=False，d=0
Output: 扩散过程产生的代价 cost

```
if limit and d ⩾ graph.limit then
    return 0
end
初始化 cost ← 0, queue ← ϕ                          ▷ 此时 tms 只有一个云节点，云节点和所有节点有连边
vertex ← graph.getVertex(nodeid)
 tms.addVertex(vertex)                                ▷ 将当前节点加入 tms 中
 tms.setR(vertex,vertex.isR)
for ovid, w in vertex.adjcent do
    ov ← graph.getVertex(ovid)
    if ov.isvisit = False and ov.isR =True then
        ov.isviste ← True                             ▷ 将节点放入 tms
        tms.add(Vertex(ov))
        queueR.put(ovid)                              ▷ 将节点放入队列
    end
    while queue.empty = False                         ▷ 递归扩散，尽可能访问多的访问 r 节点 do
        next ← queue.get()
        tmpcost+ = BEFToNeighbor(graph, mst, next, limit, d + 1)▷ 只有当 limit=True 时，d 才会有效
    end
end
return cost
```

5 复现实验

5.1 实验环境搭建

原文使用的实验集是 EUA[8], 复现时依照原文所给的节点个数，边稠密度等参数利用 ER(N,P) 对点进行概率采样连边得到[9]小规模数据集来对比 5 种算法的性能. 给定节点个数和图种边的稠密度，生成一个随机采样图. 进一步给定 r 节点占总结点数的比例即可构造一个 EDD 问题模型图. 假设输入节点个数为 N=10, 边的稠密度为 d=1.8，通过转化计算得到 p ,$p = d * N/C_N^2 = d * 2/(V-1)$,G(N,P) 模型采样后得到结果如图 12所示, 进一步输入 R 节点比例 $R = 0.6$, 得到一个 EDD 图如 13所示.

python 复现文章 set1 实验部分并将复现结果与原文对比, 实验数据集详细数如表 1所示.Set1.1 测试节点个数对算法的影响，Set1.2 测试的是边个数的影响，Ser1.3,Set1.4 分别观察 r 节点个数和 dlimit 的影响. 除此之外，复现时还补充了针对于基于随机模型 G(N,M) 生成的图上的测试. 结果发现了一些很有趣的现象. 体现了贪心之经典之处，也说明了 EDDA 仍旧可改进的点.

表 1: EUA 数据构造小规模数据集

序号	V 总边缘节点个数	D 图边的密度	Rationr 节点占 V 比例	Dlimit 延迟限制跳数
Set1.1	**10，15，…,35**	1.0	0.6	2
Set1.2	20	**1.0,1.2,...,2.0**	0.6	2
Set1.3	20	1.0	**0.2,0.4,...,1.0**	2
Set1.4	20	1.0	0.6	**1,2,...5**

[8]https://github.com/swinedge/eua-dataset

[9]https://wiki.swarma.org/index.php/ER 随机图模型

5.2 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析. 观察每种算法的 cost 值和运行时间 time(s). 衡量指标：平均误差率 ε 2，衡量近似算法的 cost 和最佳策略的差距，值越大说明偏离最优策略越大，策略传输代价越差. 幅度 η 3，衡量算法针针对集中规模下的策略代价的变化，幅度越大说明算法得到的策略通用性越差.

$$\varepsilon = \frac{(cost_{other} - cost_{eddip})}{cost_{eddip}} * 100\% \tag{2}$$

$$\eta = \frac{maxcst - mincost}{mincost} * 100\% \tag{3}$$

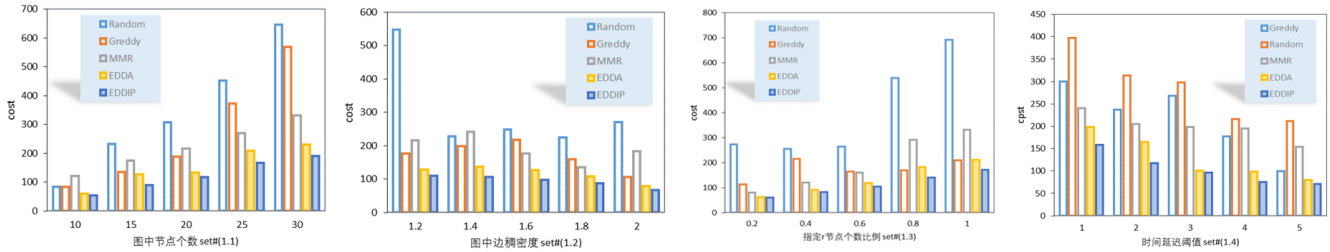
5.2.1 基于原文的实验测试

Cost 值的变化趋势 (如图 15) 和原论文基本一致，但具体值相差较大，原因可能是输入数据图和重复实验次数 (论文重复 100 次，复现时仅重复 10 次) 不同.

运行时间 t 变化趋势 (如图 16) 和原论文基本趋势一致 EDDIP>MMR>EDDA>Greedy Random，但相差较大，原因可能是编程语言以及计时方式不同 (无源码). 核心算法 EDDA 的复现误差率最低为 0.03，因此论文提出的 EDDA 算法复现效果高达 97%.

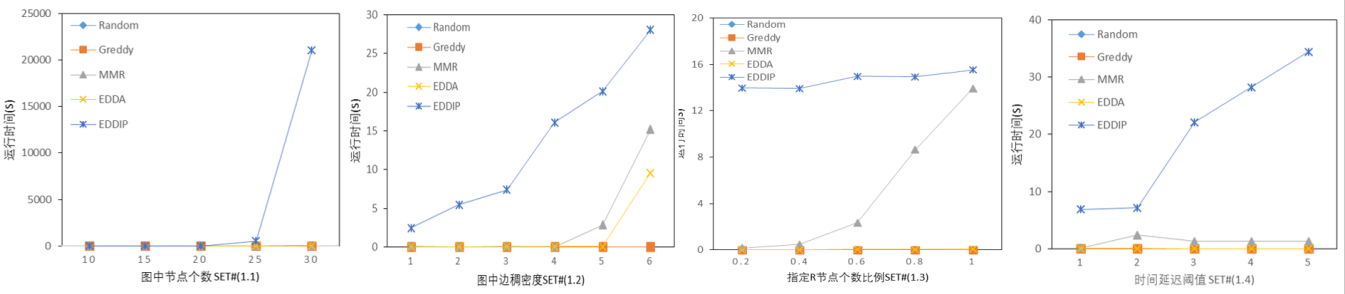
表 2: 复现和原文对比

近似算法	原文 set1.1 η	复现 set1.1η	原文 set1.2ε	复现 set1.2ε	原文 set1.3ε	复现 set1.3 ε	原文 set1.4 ε	复现 set1.4 ε	复现整体误差
EDDIP	227.18%	174%	-	-	-	-	-	-	0.11
EDDA	185.28%	**174.38%**	21.38%	**22.01%**	16.09%	**16.29%**	15.56%	**16.02%**	**0.03**
MMR	223.08%	289.83%	31.97%	30.54%	32.10%	39.09%	28.58%	32.79%	0.17
Greedy	226.62%	575.37%	40.43%	56.50%	36.10%	56.30%	29.18%	34.84%	0.21
Random	297.52%	1030.75%	53.17%	73.76%	68.06%	78.17%	48.15%	51.53%	0.38



(a)cost Vs V(set#1.1) (b)cost Vs D(set#1.2) (c)cost Vs Ration(set#1.3) (d)cost Vs dlimit(set#1.4)

图 15: 复现算法基于传输代价的对比图



(a)time Vs V(set#1.1) (b)time Vs D(set#1.2) (c)time Vs Ration(set#1.3) (d)time Vs dlimit(set#1.4)

图 16: 复现算法基于运行时间的对比图

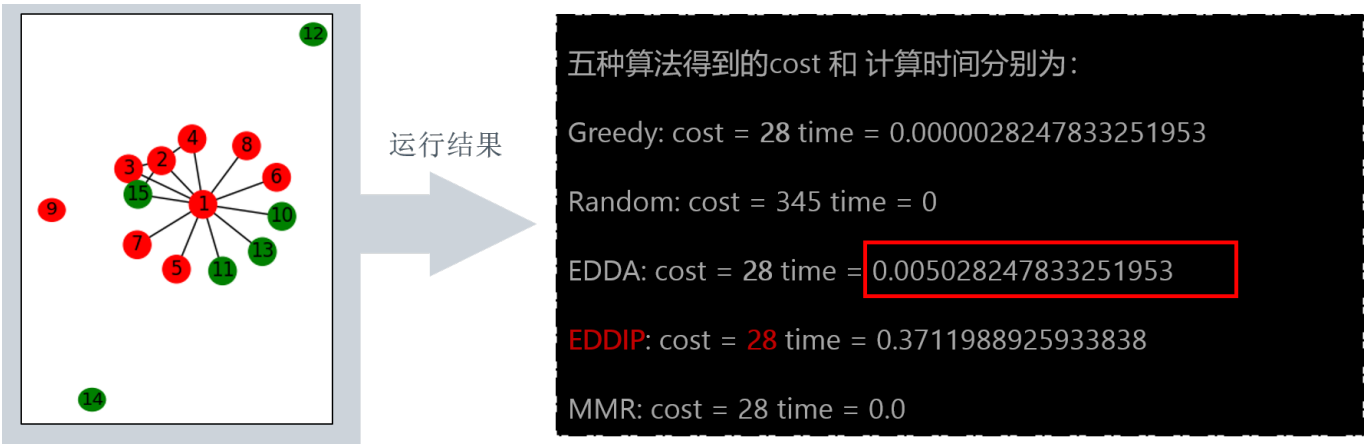


图 17: 补充实验的一个案例

5.2.2 补充实验案例

针对 ER 随即图，利用 G(N,M) 模型[10]生成的随机图, 即给定节点个数 N 和指定边条数 M 生成的随即图，经过多次测试发现，EDDA 算法容易在一下两种场景下失效:

- 场景 1：r 节点基本是图中度比较大的几个节点（Greedy 基本最优).
- 场景 2：一个比较大度的 r 节点直连到多个其他 r 节点的情况普遍（MMR 基本最优).

如图 17所示，这种场景下是满足 1、2 情况的，可以看到此时 EDDA 运行时间比 Greedy、MMR 策略长得多且代价无提升（三种策略基本能找到全局最优).

6 总结与展望

针对于 horizon 云场景下的 EDD 问题，本文提出的 EDD-A 近似算法可以在多项式时间内找到一个可行的近似解，这大大推动了类似 horizon 等的 VR 协作平台的发展. 在复现后我对随即图进行测试发现，本文 EDD-A 算法仍存在不足.

- 首先是 EDD-A 算法适用于某一特定场景，当图中度比较高节点几乎在 R 集合中时，EDD-A 几乎没有优越性，因为根据节点度选取分发节点的贪心策略可以很快速拿到最优解. 所以 EDD-A 针对图中节点度比较大的节点不宜集中在 $\mathcal{R}$ 集合的情况表现一般.
- 另一个点是，原文用特定的延迟和成本模型来解释每个实验的评估结果.

文中提到了原文工作的主要威胁是 EDD-IP 和 EDD-A 是否可以推广并应用于边缘计算环境中的其他应用场景，虽然原文实验改变了参数，以改变 EDD 问题的大小和复杂性增加评估的代表性和全面性，但只是减轻了这一缺陷，仍旧不能保证 EDD-A 的普适性. 因此，未来可能需要考虑更多条件来调整分发策略，甚至可以考虑加入机器学习等方法逼近最优策略.

[10]https://zhuanlan.zhihu.com/p/512302313

参考文献

[1] SHI W, CAO J, ZHANG Q, et al. Edge computing: Vision and challenges[J]. IEEE internet of things journal, 2016, 3(5): 637-646.

[2] HE Q, CUI G, ZHANG X, et al. A game-theoretical approach for user allocation in edge computing environment[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(3): 515-529.

[3] DROLIA U, GUO K, TAN J, et al. Cachier: Edge-caching for recognition applications[C]//2017 IEEE 37th international conference on distributed computing systems (ICDCS). 2017: 276-286.

[4] ZHANG X, ZHU Q. Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G edge computing mobile wireless networks[J]. IEEE Wireless Communications, 2018, 25(3): 12-20.

[5] ZHANG K, LENG S, HE Y, et al. Cooperative content caching in 5G networks with mobile edge computing[J]. IEEE Wireless Communications, 2018, 25(3): 80-87.

[6] CAO X, ZHANG J, POOR H V. An optimal auction mechanism for mobile edge caching[C]//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). 2018: 388-399.

[7] HALALAI R, FELBER P, KERMARREC A M, et al. Agar: A caching system for erasure-coded data[C]//2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). 2017: 23-33.

[8] YAO H, BAI C, XIONG M, et al. Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing[J]. Concurrency and Computation: Practice and Experience, 2017, 29(16): e3975.

[9] YIN H, ZHANG X, LIU H H, et al. Edge provisioning with flexible server placement[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(4): 1031-1045.

[10] LAI P, HE Q, ABDELRAZEK M, et al. Optimal edge user allocation in edge computing with variable sized vector bin packing[C]//International Conference on Service-Oriented Computing. 2018: 230-245.

[11] XU J, CHEN L, ZHOU P. Joint service caching and task offloading for mobile edge computing in dense networks[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. 2018: 207-215.

[12] WANG C, LIANG C, YU F R, et al. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing[J]. IEEE Transactions on Wireless Communications, 2017, 16(8): 4924-4938.

[13] HE T, KHAMFROUSH H, WANG S, et al. It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources[C]//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). 2018: 365-375.

[14] WANG S, URGAONKAR R, HE T, et al. Dynamic service placement for mobile micro-clouds with

predicted future costs[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(4): 1002-1016.

[15] MAHMUD R, SRIRAMA S N, RAMAMOHANARAO K, et al. Profit-aware application placement for integrated fog–cloud computing environments[J]. Journal of Parallel and Distributed Computing, 2020, 135: 177-190.

[16] BREITBACH M, SCHÄFER D, EDINGER J, et al. Context-aware data and task placement in edge computing environments[C]//2019 IEEE International Conference on Pervasive Computing and Communications (PerCom. 2019: 1-10.

[17] SHAO Y, LI C, TANG H. A data replica placement strategy for IoT workflows in collaborative edge and cloud environments[J]. Computer Networks, 2019, 148: 46-59.