基于信息新鲜度的有限容量边缘缓存更新算法设计

邹清林

学　　院：电子与信息工程学院　　专　　业：通信工程

学　　号：**180210405**　　指导教师：罗晶晶

2022 年 **6** 月

哈尔滨工业大学深圳校区

毕业设计（论文）

题　　目　基于信息新鲜度的有限容量边缘缓存更新算法设计

姓　　名　邹清林

学　　号　180210405

学　　院　电子与信息工程

专　　业　通信工程

指 导 教 师　罗晶晶

答 辩 日 期　2022 年 6 月 11 日

摘　要

随着各种多媒体设备大面积推广以及通信技术的高速迭代，互联网的用户数呈现爆发式的增长，全球总数据流量急剧增加，加重了网络负载。为了提高数据处理效率，厂商开发了边缘缓存技术，该技术利用在数据中心与用户之间使用中间存储的操作来减少不必要的下载活动从而提高缓存性能，然而采用这项技术会使得边缘节点中存储的文件的相对于传统模式更易过时。

本文研究了在一个容量有限的边缘缓存系统中的内容更新问题，创新性地综合考虑了信息的新鲜度与流行度。

首先，本文建立了在用户随机请求下的该基站内容存储的数学模型，通过添加一个基于信息年龄（AoI）的惩罚成本，从该数学模型里建立起一个关于内容更新的整数规划问题。在这之后，本文通过把该问题向一个已知的整数划分问题的约化过程证明了该问题为一个 NP-hard 难度问题，但可以通过数学规划方法解决。

为了避免直接求解计算缓慢的缺点，本文对该问题做了重新表述，通过对数学规划中列生成算法的子问题与最短路径算法（Dijkstra 算法）做映射，本文得到了一种基于列生成算法的解决方案。由于该方法得到的可能不是整数解，本文还添加了相关的舍入算法和函数，使非整数的解快速收敛得到一个整数解。

本文对设计的算法进行了仿真并且与现有的更新算法进行比较，并通过仿真结果证明该算法在系统成本上有显著优势。相比现有的更新方案，本文得到的是综合考虑了信息的新鲜度与流行度以及有限的容量的边缘缓存更新方案，实际性能更好。

关键词：边缘缓存；信息年龄；基站；最短路径算法；列生成算法

Abstract

With the large-scale promotion of various multimedia devices and the high-speed iteration of communication technology, the number of Internet users has increased explosively, and the global total data traffic has increased sharply, increasing the network load. To improve the efficiency of data processing, manufacturers have developed edge caching technology, which uses the operation of using intermediate storage between the data center and users to reduce unnecessary download activities and improve cache performance. However, using this technology will make the files stored in edge nodes more obsolete than the traditional model.

This paper studies the problem of content updating in an edge cache system with limited capacity, and creatively considers the freshness and popularity of information.

Firstly, this paper establishes a mathematical model of content stored in the base station under the random request of users. By adding a penalty cost based on information age (Aoi), an integer programming problem about the content update is established from the mathematical model. After that, by reducing the problem to a known integer partition problem, this paper proves that the problem is NP-hard, but it can be solved by mathematical programming.

To avoid the disadvantage of slow calculation of direct solution, this paper restates the problem. By mapping the subproblem of the column generation algorithm in mathematical programming with the shortest path algorithm (Dijkstra algorithm), this paper obtains a solution based on the column generation algorithm, and the amount of calculation of this solution is much less than that of the usual methods. Since the solution obtained by this method may not be an integer solution, this paper also adds relevant rounding algorithms and functions to make the non-integer solution converge quickly to an integer solution.

In this paper, the designed algorithm is simulated and compared with the existing update algorithm. The simulation results show that the algorithm has significant advantages in system cost. Compared with the existing update schemes, the edge cache update scheme, which comprehensively considers the freshness and popularity of information and the limited capacity, has better actual performance.

Keywords: edge caching, age of information, base station, Dijkstra algorithm, column generation algorithm

目 录

第 1 章　绪　论

1.1　课题背景及研究的目的和意义

近年来，我国多媒体行业以及通信行业都有着迅猛发展，随着各种多媒体设备大面积推广以及通信技术的高速迭代，一系列多媒体设备走进了人们的生活中，比如智能手机以及平板电脑。这些智能设备为人们的生活提供了便利，拉近了人与网络世界的距离。现在，人们可以通过移动平板或者笔记本电脑来进行远程视频会议，使用智能手机来观看各种视频与收听新闻。同时，多种智能设备的推广催生了各种各样的应用软件，如游戏软件、通讯软件、定位导航软件等。然而，随着智能设备的普及与用户的增多，需要发送的数据量也在高速增长，增加了通信网络的传输压力。

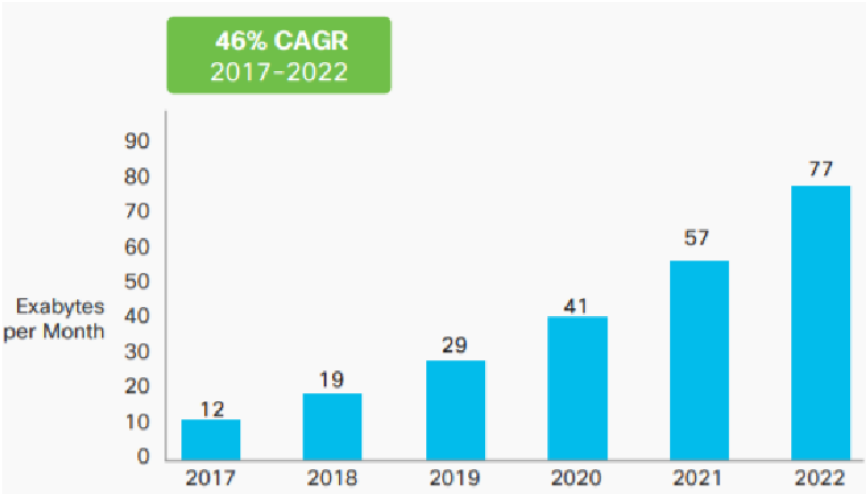


图 1-1　Cisco 公司关于全球总数据流量的统计和预测

如图 1-1 所示，Cisco 公司的一项调查显示了数据流量增长趋势移动数据流量从 2017 年至 2021 年已经增长了近 4 倍，并且预测截止 2022 年底，全球总数据流量会变成 2017 年的约 7 倍[1]。庞大的用户请求不仅给无线传输链路带来巨大的压力，也给相关厂商的服务器带来了压力[2]。目前多媒体行业常用的模式依旧是把所有的处理全部放在厂商的数据中心，由远程服务器来响应所有的用户请求。面对庞大的用户请求，这种处理方案会给远程服务器带来很大负载，大大延长了服务器的响应时间，同时远距离传输的高时延也会影响用户的体验，因此这种方法很难在大规模边缘智能设备的增长的前提下带来良好的用户体验。

为了提高用户体验从而增加自身的市场竞争力，部分厂商开始着眼于研究如何减少用户获取内容的时间。研究表明大部分的用户请求都集中在小部分的内容文

件上[3]。例如视频网站上，少数的热门视频（例如综艺类节目）占据了绝大多数的播放量，而一些较冷门的视频（如访谈类节目）则很少被观看。因此，内容提供商想到了一种方法——对于经常被访问的热门内容，如果内容提供商将这些内容储存在离用户较近的地方，就可以有效地降低用户从发出动作到得到响应的时间。这个想法推动了边缘计算以及边缘缓存技术的产生。

边缘计算（Edge Computing）技术是指通过在靠近数据源头的一侧建造一个具有计算与服务能力的数据综合处理平台来就近提供服务，从而产生更快的网络服务响应。这项技术的应用带动了一种新的缓存技术出现，即边缘缓存（Edge Caching）技术。边缘缓存是一种在传统或超大规模数据中心与访问资源的用户终端之间使用中间存储的操作来减少不必要的下载活动的缓存技术，可以提高缓存性能。采用一个数据服务器来存储庞大的数据的传统缓存系统，其在用户请求时将内容发送给用户，有成本高，网络负载大的缺点。比起传统方式，边缘缓存技术将传统的缓存方法和机制集成到基站中，将内容存储移到更靠近最终用户的地方，让网络资源更接近用户，使用更方便。实际上，不仅基站，笔记本电脑、移动设备、物联网设备和小型分支机构的文件服务器，都可以采用这种缓存机制。随着半导体技术的发展，现在较小的存储设备上可以存放更多的数据，使得各种小型基站更加普遍，为边缘缓存这项技术的发展提供了硬件的基础，未来边缘缓存技术将在数据处理方面占据更加重要的地位。

然而，边缘缓存技术会使得缓存中存储的文件的新鲜度相对传统模式较低。新鲜度是用来表征表示消息的新鲜程度的概念，刚刚产生的信息新鲜程度高，产生很久的信息新鲜程度低。在通信研究中，为了量化消息的新鲜度，采用信息年龄的概念，定义为接收的信息从生成到被接收经过的时间，如果这段时间很长，说明信息从生成到被接受经过了很久，信息的新鲜度很低，通常需要更新或者从缓存中删除。

另一种表示信息属性的概念是流行度，流行度是反映一条信息近期热度的概念。在网络服务模型中，远程服务器有数以万计的信息，而客户可能会请求到任意一条信息。被请求得更频繁的信息拥有更高的流行度，也更值得去下载到基站缓存中。

由于基站通常容量较小，应该储存流行度更高的资源，因此产生了边缘缓存的更新问题：通常情况下，信息随着时间的流逝而新鲜度下降，为了保证用户得到信息的新鲜度，我们希望更频繁地在边缘缓存中更新信息。但是由于网络资源有限，如果缓存都以较快的速率进行信息更新，将会导致网络负载较大，甚至导致网络拥塞。另外，用户请求的信息多种多样，往往超出了边缘缓存容量限制。因此，在网络资源受限的情况下，需要考虑一种基于信息新鲜度的边缘缓存更新机制。

1.2 边缘缓存国内外研究现状及分析

最初的设计方案在考虑策略时只关注了内容的流行度。2014 年的 P. Blasco 与 D. Gndz 的研究与 2015 年 Kader 的研究表明，可以通过基于学习的算法来估计内容的流行度[4-5]。2016 年，Kiskani 和 Sadjadpour 发表了一种更新策略，其中考虑了具有一定流行度的内容[6]。然而在上述研究中，内容的流行度是不随时间变化的。2018 年，Harath 在其文献中，研究了流行度在随时间变化的条件下缓存的更新策略[7]，同年，上海交通大学的 N.Zhang 等人提出了一种算法来估计随时间变化的内容流行度[8]。

对于边缘缓存更新策略的研究正随着相关的探究而更加深入，研究人员开始考虑到内容的新鲜度。在两篇文献[9-10]的研究中，两个团队开始使用信息的新鲜度，而不是内容的流行度。在这之后学术界的研究方向开始转向将内容的新鲜度和流行度结合起来考虑。在 Kam、Yates、Heydar Abad 三个团队的研究中，他们兼顾了内容的流行度和新鲜度，但是这些研究仍有不足。在 Kam 团队的研究中，忽略了从服务器下载内容的成本[11]。在 R. D. Yates 的研究中，缓存的更新能力被设置为每个时隙只能更新缓存的一个内容[12]，而实际上缓存可以更快的速度进行更新。在 M. S. Heydar Abad 的文章中，作者假设缓存容量是无限的[13]，然而实际基站往往受到缓存容量的限制。可以看出，考虑实际应用的缓存优化要同时考虑很多因素，而之前的研究未考虑全面。2020 年，Uppsala 大学的 Ahani 和 Yuan 提出了一种涵盖相对全面因素的缓存更新算法[14]，有一定指导意义。

对于如何改进现有的边缘缓存技术，已经有十数篇文献提出了相关方案。但有的研究忽略了从服务器下载内容的成本，有的研究忽略了缓存的快速更新能力，有的研究则假设有无限的缓存容量。这些研究所考虑的情形与实际应用相差甚远，实际的缓存模型不仅容量有限，而且距离服务器很远，从服务器下载内容的成本要远高于从缓存基站中下载内容的成本。要得到一个适合实际应用的边缘缓存更新策略还有待研究，比如同时考虑新鲜度与流行度、新鲜度和流行度的时变性、基站下载内容的成本、基站的容量限制等许多因素，之前的研究未考虑全面，得到的更新策略性能不理想。截至目前，还没有一种算法能同时考虑以上因素，这就使得如何找到合适的更新算法成为了边缘缓存大面积推广的关键问题。

1.3 主要研究内容

如图 1-2 所示，本文研究在缓存容量有限的情况下的边缘缓存更新机制。拟设计一种边缘缓存的更新机制，该机制不仅考虑到有限的容量，还兼顾内容的新鲜度

与流行度。通过在缓存节点上应用该机制，边缘缓存系统能够得到相对较小的网络负载。

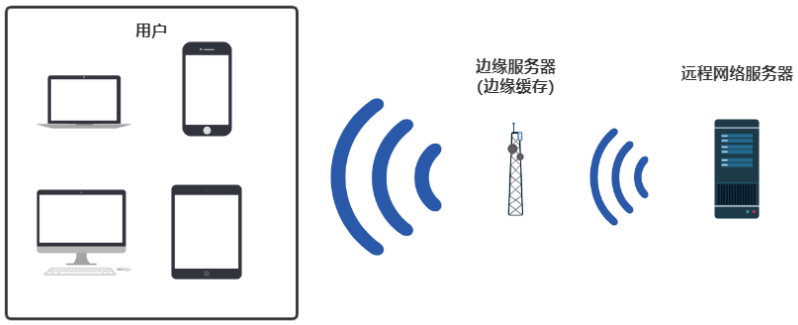


图 1-2　边缘缓存模型

本文的主要研究内容如下：

1）分析缓存容量有限的情况下的边缘缓存模型，建立在用户随机请求下的边缘缓存数学模型，研究边缘缓存基站的更新机制。在实际的边缘缓存模型中，边缘缓存更新要考虑到信息的新鲜度以及流行度，以及有限的缓存容量。该问题中存在一个优化问题，在一系列约束下进行优化，得到最优的更新策略。

2）建立优化问题，在考虑新鲜度流行度以及容量有限的缓存节点下进行优化，目标是在缓存容量约束下，通过最小化与内容下载、内容更新和 AoI 相关的内容过时惩罚成本来减轻网络的负载。

3）联合使用列生成算法与最短路径算法，添加舍入算法，得到缓存更新机制，将得到的缓存更新机制进行仿真分析并与已有算法进行性能对比。

先前的理论研究表明，缓存更新问题可以被表述为一个优化问题。具体来说，它是一个整数线性规划（Integer Linear Programming，ILP）问题，且该问题是可解的。但是，时隙数、用户数和文件数等因素的变化会造成优化问题求解时间的变化，当维数过高时，求解时间呈指数式增长。因此，本文将该问题进行重新表述，并采用了数学规划中的列生成算法来解决。本文证明了列生成算法的子问题与可以在多项式时间内求解的最短路径问题的映射关系。通过使用子问题的映射关系，我避免了直接采用数学规划方法求解带来的计算缓慢缺点，而是采用 Dijkstra 最短距离方法来求得子问题的最优解。由于列生成算法的解可能不是整数的，所以本文设计了一个舍入算法来修改该算法得到的决策，并添加了舍入函数保证快速收敛以得到满足要求的解。通过以上方法，我们可以较快地求出在给定用户数，文件数等一系列条件下的缓存最优更新策略，且数学软件的仿真数据证明了该方法在系统总成本上的优越性。相较于以往研究，本课题分析了一个容量有限的边缘缓存系统的

更新策略，推导了该更新策略的数学规划模型，综合考虑了内容新鲜度和流行度以及有限的边缘缓存容量，得到的结果更具有实际意义。

1.4 论文组织结构

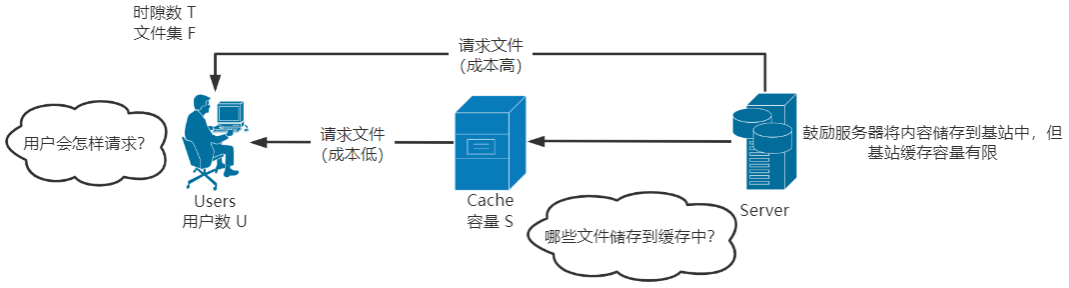
本文共四章，内容安排如下：

1) 绪论部分介绍了边缘缓存技术的研究背景以及研究的意义，对边缘缓存技术当前的国内外发展现状进行了简要概括，并且分析了已有研究的不足，同时本章从概念上介绍了本文的主要研究内容。
2) 第 2 章研究了本文研究的边缘缓存应用场景——简化的网络服务器与采用边缘缓存的基站系统。通过对该场景的分析，本章建立了用户请求的模型，缓存的更新机制，系统对用户请求的服务模型。并且，本章中发现这个边缘缓存模型中存在一个优化问题，提出了本文将要研究的优化问题模型并且证明了该问题是 NP-hard 难度的，且此问题可以被重构为一个线性规划问题来解决。
3) 第 3 章提出了基于重构的线性规划问题的最优策略求解方法。本章介绍了如何快速的求解这个问题，以避免多文件多用户下传统数学规划方法计算缓慢的问题。本文提出了采用基于数学规划理论与对偶变量的列生成法的求解方法，对其中设计到的限制子问题与子问题进行了数学表达。对于子问题，本章提出了使用 Dijkstra 最短距离算法的求解方法。此外，本章还介绍了用来修改列生成法得到的解的舍入算法，总结了整个算法的求解流程。
4) 第 4 章对算法进行了仿真测试与结果分析。通过数学软件仿真，本文将设计的算法与两种缓存更新算法进行了性能比较，从网络总成本方面战士了了设计的算法相比另外两种算法的性能的优越，并对仿真结果进行了简要分析。

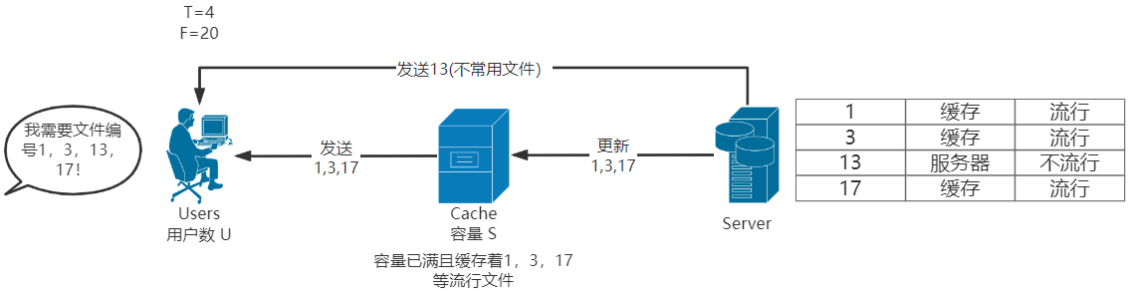
结论部分总结了前几章的工作，一方面提出本文设计的边缘缓存更新算法的不足，另一方面分析该算法可以改进的点，作为下一步研究工作的方向。

第 2 章　边缘缓存模型分析

2.1　网络模型



(a)



(b)

图 2-1　网络模型

图 2-1 是一个简化的实际服务器-边缘缓存系统。

该系统由三部分组成：一组用户集，一个边缘缓存节点与一个服务器。用表示数量为用户的集合，在每个时隙会向边缘缓存节点发出对内容的请求。先前的研究表明，用户的请求可以用数学模型表示。

边缘缓存节点（下简称缓存）可以低成本的方式将文件发送给用户，但容量有限为，因此不可以将所有内容都储存在缓存中。

内容储存服务器储存着所有文件，也具有将文件发送给用户的能力，但成本较高。因此，流行的文件应优先通过缓存发送。

所有文件构成一个数量为$|F|$的文件集合，记为F，且每个文件都有其自己的编号，此外将所有时隙的集合记为T，因此有以下三个集合：

F为文件集，文件的编号为$\{1,2,3,\ldots\ldots,|F|-1,|F|\}$，$f \in F$表示一份文件（内容）

$T=\{1,2,3,\ldots\ldots,|T|-1,|T|\}$，$t \in T$表示其中一个时隙

U为用户集，用户的编号为$\{1,2,3,\ldots\ldots,|U|-1,|U|\}$，$u \in U$表示一名用户

在分析时，只考虑单一时隙。在每个时隙，收到用户的请求后，缓存节点进行如下工作：

1) 先从缓存查询文件。
2) 如果在缓存中没有查询到，则从服务器查询。
3) 如果缓存中没有该文件，则从服务器将文件发给用户，并且将该文件写入缓存，如果缓存中有，则直接将文件发给用户，并且将文件更新。

现在采用信息年龄来量化信息的新鲜度，信息年龄 AoI 定义为缓存的中的一个文件从下载后到下一次下载（更新）之间经过的时间。设在时隙t根据用户请求，缓存中刚刚下载了文件内容f，如果该在t'时隙对缓存中的该文件进行更新，则定义内容f的信息年龄 AoI 的表达式为：

$$AoI(f) = t' - t \tag{2-1}$$

考虑如何量化内容的流行度：在内容缓存领域的相关论文显示[15]，信息的流行度服从齐夫分布（Zipf Distribution，又称 Zipf 分布），该分布是服从齐夫定律的分布。齐夫定律是由哈佛大学的语言学家 George Kingsley Zipf 于 1949 年发表的实验定律，该定律指少数英文单词在英语中的使用频率远远超过绝大多数单词的使用频率。齐夫分布同样表现在本课题中文件集F的每个内容f的请求概率上。由其公式，在本课题中对于任意一个请求，请求第f个内容的概率是：

$$p(f) = \frac{f^{-\gamma}}{\sum_{i \in F} i^{-\gamma}} \tag{2-2}$$

γ为常数,通过上式，可以得到每个请求与文件的概率关系。

2.1.1 优化问题建模

每个用户的所有请求都在每个时隙t初发生，且这些请求是随机变化的，为了进行建模，需要了解用户每个请求的内容与在哪个时隙发出请求。用户的请求集合记为R_u，$r \in R_u$表示一个用户请求。用$h(u,r)$和$o(u,r)$两个函数分别表示用户u的第r个请求与内容和时间的关系。结合两个函数，可以计算出每个文件在哪几个时隙被请求，进而计算其 AoI。用m_{tf}表示在t时隙请求内容f的用户数。

缓存还受到容量的限制，由于实际文件大小不等，讨论缓存容量与文件大小的相对大小更有意义。用l_f表示每个文件的大小（单元），设缓存大小为S，其值为所有文件的总大小的ρ倍：

$$S = \rho \sum l_f \tag{2-3}$$

使用C_s表示从服务器直接向用户提供文件的单元成本，使用C_b表示从缓存中

向用户提供文件的单元成本。

为了表示内容在缓存中存储的情况，引入两个变量，x_{tf} 与a_{tfi}。

x_{tf} 是二进制变量，表示内容f是否在t时刻存储在缓存中。当x_{tf} =1 时，括号内右边一项为 0，表示内容在缓存中，当x_{tf} =0 时，括号内左边一项为 0，表示内容不在缓存中。为了细化每个时刻的更新决策，另一个二进制变量a_{tfi}表示内容f是否在t时刻存储在缓存中且具有i的 AoI，f满足条件时该变量为 1，否则为 0。

现在考察每个时隙该系统的所有行为，涉及网络传输负载的部分有：服务器向用户发送文件，缓存向用户发送文件与服务器将内容在边缘缓存中更新，共三种行为。其中前两种都是为了满足用户的文件下载请求。如果满足所有用户的所有文件请求的行为带来的网络负载定义为一种成本，记为$C_{downlaod}$，则可以写出它的表达式:

$$C_{downlaod} = \sum_{u=1}^{U}\sum_{r=1}^{R_u} l_{h(u,r)}[\,C_b x_{o(u,r)h(u,r)} + C_s(1 - x_{o(u,r)h(u,r)})] \tag{2-4}$$

用户的下载请求还引起缓存中的内容更新，记更新成本为C_{update}。由于研究了内容的新鲜度，还需要引入一个缓存内容新鲜度下降所造成的惩罚成本，在优化时应使得该成本尽可能小，将该成本记为 AoI 成本C_{AoI}，定义内容储存成本$p_f(i)$为内容f为i的AoI与其大小l_f的乘积。。

$$C_{update} = \sum_{t=1}^{T}\sum_{f=1}^{F} l_f(C_s - C_b)\, a_{tf0} \tag{2-5}$$

$$C_{AoI} = \sum_{f=1}^{F}\sum_{t=1}^{T}\sum_{i=1}^{t=1} p_f(i) m_{tf} a_{tfi} \tag{2-6}$$

a_{tf0}表示内容刚从服务器下载，即 Aoi=0。

分析引入的两个二进制变量，x_{tf} 表示内容f是否在t时隙存储在缓存中，a_{tfi}表示内容f是否在t时隙存储在缓存中且具有i的 AoI。对一个内容f，其(x_{tf}, a_{tfi})对可以表示所有时隙中它在缓存中的情况。得到了所有(x_{tf}, a_{tfi})对，也就得到了整个系统的更新策略。我们的目标是在保证内容新鲜度的前提下降低网络负载，优化目标应为使总成本最小，该缓存优化问题表示为：

$$\min_{x,a}\ C_{downlaod} + C_{update} + \lambda C_{AoI}$$

λ为权重系数，$\lambda \in [0,1]$，λ越小，系统越不更新。λ=0 时，系统不更新缓存。

现在将上式展开并添加约束，有：

$$\min_{\boldsymbol{x},\boldsymbol{a}} \sum_{u=1}^{U}\sum_{r=1}^{R_u} l_{h(u,r)}[\,C_b x_{o(u,r)h(u,r)} + C_s(1 - x_{o(u,r)h(u,r)})] + \sum_{t=1}^{T}\sum_{f=1}^{F} l_f(C_s - C_b)\, a_{tf0} + \lambda \sum_{f=1}^{F}\sum_{t=1}^{T}\sum_{i=1}^{t=1} p_f(i) m_{tf} a_{tfi} \tag{2-7a}$$

该问题的约束条件为：

$$\begin{cases} \sum_{f=1}^{F} x_{tf}\ l_f \le S, t \in T \\ \sum_{i=0}^{i=t-1} a_{tfi} = x_{tf}, t \in T, f \in F \\ a_{tfi} \ge x_{tf} + a_{(t-1)f(i-1)} - a_{tf0} - 1, t \in T, t \ne 1 \\ a_{tfi} \le a_{(t-1)f(i-1)}, t \in T, t \ne 1 \end{cases} \tag{2-7b}$$

式(2-7b)中的第一式表示缓存容量约束，后三个式子表示 AoI 约束。后两个式子是时隙t中的内容f具有 AoI =i的条件：当且仅当内容f的AoI在从时隙 2 到时隙t中不为0，并且在时隙 $t-1$ 中具有 AoI=$i-1$。至此，我们得到了边缘缓存更新问题的数学模型，该问题是一个整数规划问题。在求解该问题之前，需要先分析此问题的难度。

2.2 问题分析

参考已有文献，本文在该部分证明了这个边缘缓存优化问题是一个 NP-hard 难度问题，但是可以通过整数规划方法求解。

该证明是基于向一个已有的整数划分问题的约化过程而建立的，该整数划分问题被在文献[16]中已经被证明为是一个 NP-Complete 问题并且可以通过整数规划方法求解。这个问题是：考察一个整数集$N = \{n_1, n_2, n_3, n_4, \dots\dots, n_{N-1}, n_N\}$，如何将该集$N$分为不相交的两组，$N_1$和$N_2$，且令每组都具有相同的总和。

过程如下：假设$S = 0.5 * \sum_{i=1}^{N} n_i$,令$N$为文件集$F = \{1,2,3, \dots\dots, F-1, F\}$，对应的$n_f$等于每一个文件$f$的大小$l_f$，且$T = 1$，系统始终在第一时隙，只有文件的下载决策而不涉及更新决策。同时假定所有用户的请求也只在$T = 1$时隙产生，即$o(u,r) = 1$，$u \in U, r \in R_u$。令从服务器直接向用户提供文件的单元成本C_s=2,从缓存中向用户提供文件的单元成本C_b=1，每个文件在该时隙都被请求 2 次。

在上述参数下，如果一个文件f被存储在缓存中，相比不存储在缓存中，该决策为总系统节约的开销就是不存储该文件需要下载成本减去存储该文件的成本，

即$2l_f + l_f - 2 * l_f = l_f$,也就是存储一个文件可以节约$l_f$的成本。那么问题就变为在缓存大小$S=0.5 * \sum_{i=1}^{N} n_i$的条件下，能否获得最大节约成本$0.5 * \sum_{i=1}^{N} n_i$，这个问题的解也是划分问题的解。通过这种方式，边缘缓存更新问题被约化为一个 NP-Complete 问题，因此该问题是 NP-hard 难度的。

为了方便求解，下文对问题进行重新表述。

本文定义决策对$(\boldsymbol{x_f}, \boldsymbol{a_f})$表示节点对文件的一个决策，对于一个内容$f$，考察它从0到时隙$T$里所有的决策对$(\boldsymbol{x_f}, \boldsymbol{a_f})$

$$\boldsymbol{x_f} = \left[x_{1f}, x_{2f}, x_{3f}, x_{4f}, \ldots\ldots, x_{Tf}\right]' \tag{2-8}$$

$$\boldsymbol{a_f} = \left[a_{1f0}, a_{2f0}, \ldots\ldots, a_{Tf(t-1)}\right]' \tag{2-9}$$

在一个时隙，对于文件f根据两个二进制变量的意义，一组$(\boldsymbol{x_f}, \boldsymbol{a_f})$可能有三种情况：（0，0），（1，0）和（1，1），这三种决策分别有其含义：（0，0）表示该内容不在缓存中，代表删去该内容或者该时隙没有请求，（1，0）表示继续在缓存中存储该内容但不更新，（1，1）表示在缓存中更新该内容。每个时隙有 3 个这样的决策可能，那么T个时隙内就总共有3^T个这样的二进制变量对的可能组合。这3^T个决策变量组合包括了t从1到T，文件f的所有缓存情况。某一个时隙t，对于内容f的决策可能是不提供该内容或者将该内容从缓存中删去（腾出空间给更重要的内容），可能是内容存储并不更新，或者更新内容，每一种可能的组合都是3^T个组合中的一组，也就是说，这3^T个变量对包含着该内容f的所有缓存可能。

为了重构问题，定义一个索引集$K = \{1,2,3,4, \ldots\ldots, 3^T\}$,表示所有的决策变量$(\boldsymbol{x_f}, \boldsymbol{a_f})$对组合。每一个文件的一种在所有时隙内的$(\boldsymbol{x_f}, \boldsymbol{a_f})$对组合为该索引集$K$中对应的一列，那么$k \in K$表示一种可能的解。

对于内容f，其3^T个决策可能中只有一种是实际发生的，用二进制变量w_{fk}来表示其选择情况，当且仅当内容f的决策选择了第k种可能时，该变量为 1，其余情况为 0。通过联合使用w_{fk}和K，仅使用两个变量就可以表示内容f的所有可能选择策略，做到了简化问题，且有利于下一章的算法构建。

现在使用w_{fk}来将问题简化并重构，设在所有时隙T内，一个文件f在更新策略为k下的总花费为C_{fk}，则可以写出C_{fk}的表达式为式（2-10）：

$$C_{fk} = \sum_{t=1}^{T} l_f m_{tf} [C_b x_{tf}^k + C_s(1 - x_{tf}^k)] + \sum_{t=1}^{T} l_f (C_s - C_b)\, a_{tf0}^k + \lambda \sum_{t=1}^{T} \sum_{i=1}^{t=1} p_f(i) m_{tf} a_{tfi}^k \tag{2-10}$$

那么现在目标问题被简化为：

$$\min_{w} \ C_{fk}\, w_{fk} \tag{2-11}$$

约束为：

$$\begin{cases} \sum_{f\in F}\sum_{k\in K} l_f\, x_{tf}^{k} w_{fk} \le S, \quad t \in T \\ \sum_{k\in K} w_{fk} = 1, \quad f \in F \\ w_{fk} \in \{0,1\}, f \in F, k \in K \end{cases} \tag{2-12}$$

表示缓存容量约束与二元变量约束。

2.3 本章小结

本章主要通过分析边缘缓存的实际应用场景，构建了一个简化的采用边缘缓存的基站的系统，并且参考广泛使用的系统对用户请求的服务机制，对这个系统进行了数学。提出了一个数学规划下的边缘缓存更新的优化问题，也即是求解建模缓存的更新机制，通过将此问题向一个 NP-Complete 难度的整数划分问题的约化过程，本文证明了提出的边缘缓存优化问题是 NP-hard 难度的，且此问题可以作为一个整数线性规划问题来解决。之后，通过定义决策变量对以及利用其可能的所有组合，本文换用另一种表示文件更新策略的方式，对原问题进行了重构，减少了线性规划的目标变量的规模，以方便后续算法的设计。

第 3 章　缓存更新机制研究

3.1　引言

上文表明，边缘缓存的问题可以被重构为一个简化的 ILP 问题。该问题可以采用已有的数学求解器求解，但它需要大量的计算时间。求解该问题有两个难点。首先，在原问题中，目标w为$f \times K$维的矩阵，其中K=3^T是时隙$|T|$的幂函数，随着时隙数增长，K将会变得巨大。其次，C_{fk}的计算涉及内容f从0到时隙T里所有的决策对(x_f, a_f)，计算量很大，求解时间很长。

求解整数规划问题的方法有很多，通常使用分枝定界法、割平面法、穷举法等方法，但本问题中变量数很多，上述算法都是从所有可能的变量开始规划，需要的时间相对较长，不适用于该问题，需要找到一种在变量很多的线性规划问题下也可以具有快速计算能力的算法。本文需要找到一种算法，从小规模的问题出发，不需要一开始就得到大量的决策组合，而是逐渐找到可能的解，可以节省许多时间。因此，本文采用基于单纯型法提出的列生成算法。

列生成算法（Column Generation Algorithm，CGA）是一种用于高效求解大规模线性优化问题的算法，其理论基础由 Danzig 等人于 1960 年提出。从本质上说，列生成算法是单纯形法的一种形式，该算法的思想就是从很小的变量出发不断迭代直到得到最优解，基于当前列的子集，构造一个限制主问题进行求解，限制主问题的变量数会逐渐添加，条件是新的候选方案可以改善限制主问题当前最优解。与单纯形法相比，列生成方法并不同时处理所有的候选方案。

目前，该算法已被用于求解很多著名的 NP-hard 问题，如机组人员调度问题、车辆路径问题、切割问题、单资源工厂选址问题等。此外，由于列生成算法得不到整数解，我又设计了一个舍入算法来在每个时隙上修改获得的决策，直到得到整数解，程序框图如下。

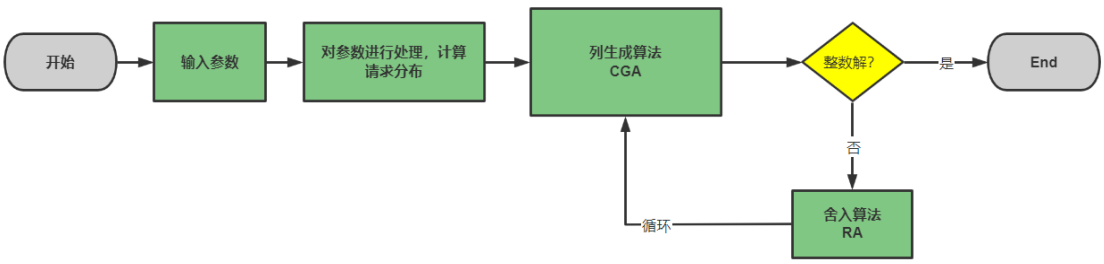


图 3-1　缓存更新机制求解程序框图

3.2　列生成算法设计

CGA 先把原问题 MP（Master Problem，MP）简化到一个规模更小的限制主问题 RMP（Restricted Master Problem，RMP）下，在 RMP 上求最优解，但是此时求得的最优解只是对于 RMP 问题的，并不是 MP 问题的最优解。我们需要通过子问题（Subproblem, SP）并计算其 reduced cost 去检验在那些未被考虑的变量。如果有对 RMP 问题有改进的，那么就把对应的列加入 RMP 问题。经过反复的迭代，直到找不到可以改进 RMP 的变量，此时得到原问题 MP 的最优解。

在求解该线性规划问题中，CGA 算法的优点得以体现：一方面，在$K' \in K$的子集中，K'可以设置的比较小，能快速得到初始解。另一方面，在初始化时如果对于所有的$f \in F$,将其所有$(\boldsymbol{x_f}, \boldsymbol{a_f})$都设为（0，0），则$C_{fk}$=$\sum_{t=1}^{T} l_f m_{tf}\, C_s$，在这种情况下 C_{fk}为一个容易求得的常数，算法能从一个简单的初始情况开始迭代。由于 CGA 算法是迭代的，在迭代循环中添加更优的解，因此总能找到最优解，避免了上述难点。

算法详细思路如下：

列生成算法 Column Generation Algorithm

1: 输入变量：$U, F, T, C_{s,} C_b, \lambda, \rho, \gamma, l_f$, S, 计算$h(u,r)$和$o(u,r)$
2: 初始化：
对于所有的$f \in F$，令其$(\boldsymbol{x_f}, \boldsymbol{a_f}) = 0$　设置标志变量 stop←0
3: **while** stop=0 **do**
4: 求解 RMP 问题
5: stop←0
6: **for** $f \in F$ **do**
7: 求解 SP 问题,得到$(\boldsymbol{x}, \boldsymbol{a})$
8: 计算 reduced cost
9: **if** reduced cost < 0
10: 向 RMP 中加入新的$(\boldsymbol{x}, \boldsymbol{a})$
11： stop←1

3.2.1 RMP 问题

通过原问题K矩阵的子矩阵K'矩阵来构造 RMP 问题：

RMP 问题表示如式（3-1）：

$$\min_{w} \; C_{fk} w_{fk} \tag{3-1a}$$

约束为：

$$\begin{cases} \sum_{f\in F}\sum_{k\in K'} l_f\, x_{tf}^{k} w_{fk} \le S, \quad t \in T \\ \sum_{k\in K} w_{fk} = 1, \quad f \in F \\ 0 \le\ w_{fk} \le 1\,, f \in F, k \in K' \end{cases} \tag{3-1b}$$

由于该算法中没有整数约束，因此w_{fk}可以得到分数结果。在初始情况下，C_{fk}为一个常数，计算非常简单。

3.3.2 SP 问题

由于 CGA 算法是迭代的，每次在 RMP 问题下求得解$\boldsymbol{w}^*$后，都需要检查$\boldsymbol{w}^*$是否是 RMP 的最优解。

由线性规划理论可知，该过程可以通过为$f \in F$ 的每个内容找到具有最小降低成本（Reduced Cost，RC，记为ΔC）的决策组$(\boldsymbol{x_f}, \boldsymbol{a_f})$来做到。做法是通过计算一个决策组合的$\Delta C$，来判断该策略是否对现有的方案有改进作用：如果$\Delta C$是负值，那说明该策略对我们的方案是有改进的，把该策略添加到 RMP 问题中，即将具有负的降低成本的决策组$(\boldsymbol{x_f}, \boldsymbol{a_f})$添加到 RMP 问题的变量中；如果$\Delta C$是正值或者为零，说明该策略对更新方案是没有改进，就不考虑它。对于得到的缓存更新方案，如果所有这些值都是非负的，则当前解决方案是最优的。否则说明还需要迭代，直到找不到对原方案有更新的新决策。为了完成这个过程，需要构建一个 SP 问题。

SP 问题使用对偶变量来构建,定义上式中与$\sum_{f\in F}\sum_{k\in K'} l_f\, x_{tf}^{k} w_{fk} \le S$与$\sum_{k\in K} w_{fk} = 1$两式对应的最优对偶变量为$\pi$和$\beta$：

$$\pi = [\pi_1, \pi_2\,, \pi_3, \ldots\ldots, \pi_T\ ]' \text{ 与} \beta = [\beta_1, \beta_2, \ldots\ldots, \beta_F]'$$

通过调用数学规划标准求解器的函数，可以在求 RMP 问题时直接求出π和β两组对偶变量。由线性规划的对偶问题相关内容可知，对于一组内容f，决定是否有新变量加入 RMP 问题的ΔC为

$$\Delta C = C_f - \sum_{t=1}^{T} l_f \pi_t\, x_{tf} - \ \beta_f \tag{3-2}$$

$$C_f = \sum_{t=1}^{T} l_f m_{tf}[C_b x_{tf} + C_s(1 - x_{tf})] + \sum_{t=1}^{T} l_f (C_s - C_b)\, a_{tf0} + \lambda \sum_{t=1}^{T}\sum_{i=1}^{t=1} p_f(i) m_{tf} a_{tfi} \tag{3-3}$$

用$(\boldsymbol{x^*}_{\boldsymbol{f}}, \boldsymbol{a^*}_{\boldsymbol{f}})$表示子问题$SP_f$的最优解。如果$(\boldsymbol{x^*}_{\boldsymbol{f}}, \boldsymbol{a^*}_{\boldsymbol{f}})$的降低成本为负，就将其添

加到 K'中。

SP 问题表示如式（3-4）：

$$\min_{(x^*_f, a^*_f)} C_f - \sum_{t=1}^{T} l_f \pi_t x_{tf} - \beta_f \tag{3-4a}$$

约束为：

$$\begin{cases} \sum_{i=0}^{t-1} a_{tfi} = x_{tf}, t \in T, f \in F \\ a_{tfi} \geq x_{tf} + a_{(t-1)f(i-1)} - a_{tf0} - 1, \quad t \in T, t \neq 1 \\ a_{tfi} \leq a_{(t-1)f(i-1)}, \end{cases} \tag{3-4b}$$

通过上式可以看到，SP 问题所求的优化目标是所有的决策变量组，同样无法避免求解时间很长的问题，下文讲述如何采用迪杰斯特拉算法的方法来解决 SP 问题。

3.3.3 SP 问题与 Dijkstra 算法

迪杰斯特拉算法（Dijkstra）在 1959 年由荷兰计算机科学家 Dijkstra 提出。该算法使用找到从一个顶点到其余各顶点的最短路径的方式来解决有向无环路非负权图中的最短路径问题。

SP 问题可以被转化为最短路径问题，并且可以通过 Dijkstra 算法来求解，具体映射过程如下[14]:Dijkstra 算法需要有向无环图，考虑每一个$f \in F$，首先构建一张关于其的有向无环图，找到从起点到重点的最短路径的过程就等效于求解关于f的 SP 问题，SP_f问题。

考察一个SP_f问题，其目标函数为$C_f - \sum_{t=1}^{T} l_f \pi_t x_{tf} - \beta_f$，可以先将$x_{tf}$代入之后把该目标函数重写为如式（3-5）的形式：

$$\sum_{t=1}^{T} l_f m_{tf} C_s + \sum_{t=1}^{T} \left\{ l_f (C_s - C_b) a_{tf0} + \sum_{i=1}^{t-1} p_f(i) m_{tf} a_{tfi} - \left[l_f m_{tf} (C_s - C_b) - l_f \pi_t \right] x_{tf} \right\} - \beta_f \tag{3-5}$$

除了$-\beta_f$，式中还有如下四部分：

$$C = \sum_{t=1}^{T} l_f m_{tf} C_s \tag{3-6a}$$

为所有时隙内请求内容的所有用户通过服务器下载内容f的总成本。

$$c_1 = l_f (C_s - C_b) \tag{3-6b}$$

表示从服务器到缓存的下载成本。

$$v_{it} = p_f(i)m_{tf} \tag{3-6c}$$

表示m_{tf}位用户在时隙 t 请求内容 f，并且内容具有 AoI = i时的成本。

$$g_t = l_f m_{tf}(C_s - C_b) - l_f \pi_t \tag{3-6d}$$

表示由于存储内容 f 而导致的 C 成本的减少值。

依照内容储存的关系，可以将缓存以如下方式表示：起点为S终点为D，并依照每个时隙来生成每个时隙的节点列。节点V_{00}表示每个文件的x_0=0,即初始状态不在缓存中。首先，每种决策都具有成本β_f，因此最后一个时隙的节点列到终点D的弧权重都为$\beta_f(\beta_f>0)$。其次，对于每个时隙t，生成 t+1 个相对应的节点，这些节点一共分为 3 类，节点V_{t0}表示内容不在缓存中，V_{t1}^{i} 表示内容在缓存中且具有i的 AoI，即i=0 时内容在该时隙被更新，其余情况表示内容仅被储存在缓存中但不更新。

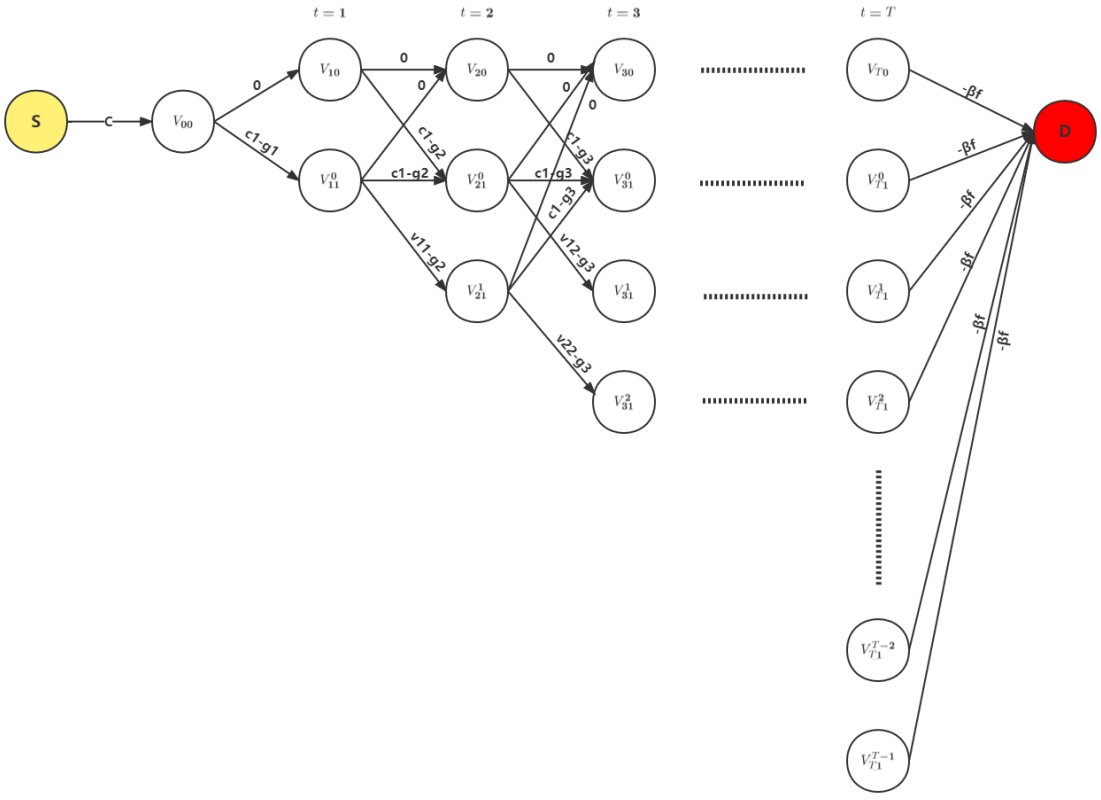


图 3-2　Dijkstra 算法与 SP 问题的映射

如图 3-2 所示，每两个节点之间都有一条权重大于等于 0 的弧。从节点S到节点V_{00}有一条权重为C的弧。对于每个V_{t0}节点，有两条指向下一时隙t+1 的弧，一条指向$V_{(t+1)0}$，这意味着内容不存储在下一个时隙中且权重为0，另一条指向$V_{(t+1)1}^{0}$，权重为$c_1 - g_t$，表示内容在下一个时隙下载到缓存中。对于每个V_{t1}^{i} 节点，有三条

指向下一时隙的弧，一条指向$V_{(t+1)0}$，权重为0，表示下一时隙删掉该内容；一条指向$V^{0}_{(t+1)1}$，权重为$c_1 - g_t$，表示内容在下一时隙被更新；最后一条指向$V^{i+1}_{(t+1)1}$，说明内容保持不变，其 AoI 增加一个单位，权重为$v_{(i+1)(t+1)} - g_{t+1}$。

通过该方式来构造 Dijkstra 节点图，那么每一种 SP 问题的解都可以直接映射到具有相同目标函数的从源到目的地的路径。相反，给定一条路径，同样可以找到一种 SP 问题的解决方案与之对应。更准确地说，假如得到了一条最短路径，对于时隙 t，如果路径决定经过节点V_{t0}，则设置x_{tf}= 0。如果路径经过V^{i}_{t1}，我们设置x_{tf}=1 和a_{tfi}=1。在上文中已经表明，如果知道了内容在每个时隙的存储情况，就可以反推出其储存决策，因此使用 Dijkstra 算法可以解决 SP 问题，且由于该算法本身的计算性能，问题可以在多项式时间内解决[17]。此外，通过采用该算法，由于上文已经计算了每种方案的总花费C_f，可以不进行SP_f问题中ΔC的计算，直接比较两种方案的总花费也能得到相同的结果。

3.3 舍入算法设计

考虑到列生成算法只能得到最优解，但不能保证得到整数解，还需要设计一个舍入算法，将其与 CGA 算法一起使用来得到线性规划的最优整数解。在 CGA 算法得到方案之后，对该缓存更新方案进行检验，如果得到的解并不是一个整数解，则采用 RA 算法进行舍入。RA 算法根据每个文件被请求的可能性在时隙上反复修复内容的缓存决策，直到构造出整数解。

在时隙 t是否缓存内容 f 的缓存决策基于值 z_{tf} 确定，该值表示在时隙 t 中存储内容 f 的可能性有多大。定义为

$$z_{tf} = \sum_{k \in K'} x^{k}_{tf} w_{fk} \tag{3-7}$$

z_{tf}与w_{fk}存在着对应关系：对于内容$f \in F$，$\boldsymbol{z}_f = [\, z_{1f}\,, z_{2f}\,, \ldots, z_{tf}]$ 是二元矩阵的充要条件是对于任何k, w_{fk}都是二元的[14]。根据得到的 z_{tf}来修改决策，如 z_{tf}=1，说明在缓存决策中一定会在时隙t存储内容f，那就把对应要修改的表示决策的相应变量中(x_{tf}, a_{tfi})中的x置 1，并且把所有不符合此缓存决定的决策丢弃。在此过程之后我们找到 z_{tf}中的最接近 0 或 1 的元素并进行舍入。根据舍入结果，确定缓存决策并丢弃不符合的列。

由于之前的删除操作可能会删除所有生成的内容f的决策组（列），RA 算法需要在决策中添加一组决策，该组决策在 x 值固定为 1 的时隙中具有$\boldsymbol{x_f}$=1，而在其他时隙中有$\boldsymbol{x_f}$=0 来确保所有内容都会在最终决策中被使用到。RA 算法的输出为

修改后的决策组，由于在循环中，因此之后程序会继续调用 CGA 来求解w_{fk}，直到得到整数解。

算法详细思路如下：

舍入算法 Rounding Algorithm

1: 输入变量：$K, U, F, T,\ l_f, S, (\boldsymbol{x_f}, \boldsymbol{a_f})$
计算 z_{tf}=$\sum_{k\in K'} x_{tf}^k w_{fk}$
定义$\bar{S}$=S，$\bar{F}$=[ ]
2: 如果 z_{tf}=1，对应的变量对(x_{tf}, a_{tfi})中的x置 1
3: 如果 z_{tf}=0，对应的变量对(x_{tf}, a_{tfi})中的x置 0
4: 找到 z_{tf}中的最接近 0 的元素，$z_{lb} = min\{z_{tf} | 0 < z_{tf} < 1\}$
及其对应的t_{lb}, f_{lb}= $argmin\{z_{tf} | 0 < z_{tf} < 1\}$
5: 找到 z_{tf}中的最接近 1 的元素，$z_{ub} = max\{z_{tf} | 0 < z_{tf} < 1\}$
及其对应的t_{ub}, f_{ub}= $argmax\{z_{tf} | 0 < z_{tf} < 1\}$
6: **if** $z_{lb} < 1 - z_{ub}$
7: 对应的变量$x_{t_{lb}, f_{lb}}$置 0
8: **else if** $(l_{f_{ub}} < \bar{S})$
对应的变量$x_{t_{ub}, f_{ub}}$置 1
9: **else** $x_{t_{ub}, f_{ub}}$置 0
10: **for** $f \in F$ **do**
如果文件f的某个时隙的x_{tf}被置 1，且其他时隙x_{tf}为 0,则在该时隙添加一个决策$(\boldsymbol{x_f} = \boldsymbol{1}, \boldsymbol{a_f} = \boldsymbol{0})$来请求该内容$f$，防止该文件在后续决策中被误删除
11: **for** $t \in T$ **do**
如果x_{tf}被置为 1，$\bar{F}$就添加对应的f
$\bar{S}$=S-$\sum \overline{l_f}$，且$\overline{l_f}$对应$\bar{F}$中的文件大小
12: **for** $f \in F$
if $l_f < \bar{S}$ 对应的变量x_{tf}置 0
13: 从得到的(x_{tf}, a_{tfi})来生成新的决策组$(\boldsymbol{x_f}, \boldsymbol{a_f})$添加入 CGA 问题
14: 记录迭代次数，当达到一定值时，根据迭代次数来进行决策舍入

3.4 总程序设计

程序设计总体思路如下，首先输入参数，包括时间数T，用户数U，文件数F以

及相关系数等。每个文件的大小l_f随机产生。接下来对输入参数进行处理，计算缓存容量S以及用户的内容请求模型，由于产生的请求模型是完全随机的，因此可能产生无法进行优化的模型，需要生成新的一组请求模型。

上文 CGA 已经提到，程序主要处理对象是决策对$(\boldsymbol{x_f}, \boldsymbol{a_f})$，因此下一步需要生成最初的一列决策对组合$(\boldsymbol{x_f}, \boldsymbol{a_f})=(\boldsymbol{0}, \boldsymbol{0})$为一系列$(x_f, a_f)$对，共$T$个时隙，每个时隙对应文件数生成$F$个决策对，共$T\times F$个。

为了后续求解方便，采用变量k来记录迭代次数，初始化时k置 1，之后将最初的决策组合和生成的用户请求模型传入 CGA 进行求解，CGA 中调用数学规划的模型求解器。在 CGA 中首先按求解器的标准形式经过该过程之后新的决策会作为新的列被添加到$(\boldsymbol{x_f}, \boldsymbol{a_f})$中，每次迭代都会添加 1 列并令$k$+1。CGA 算法得到的解就是$w_{fk}$，首先进行检验是否是整数解，如果不是就采用 RA 算法进行优化。

然而，在实际仿真中，足够长的时间下 RA 算法的确可以收敛到整数解。实际上，对于一个文件f，在前十几次迭代中 RA 算法可以快速地将绝大部分不需要的决策舍去，而之后的循环中该算法会不断地在剩下的两个可能的决策中优化，直到一个决策被优化到 0，考虑到 RA 算法和 CGA 算法的联合使用可能会造成收敛时间长，且几十次循环就可以看到最终决策的优化趋势，本文使用一个简单的舍入函数来节省时间，在迭代次数足够多能看到内容决策的更新趋势后，该舍入函数可以直接针对两个可能决策进行选择，从而缩短算法的运行时间，提高效率。

另外，为了避免 Dijkstra 算法中产生负的权值的问题，所有会产生负的权值的弧，程序中都将其权重设置为 0，由于使用 Dijkstra 算法的目的是得到最优路径而不是最短距离，因此该方法不会对结果产生影响

算法详细思路如下：

基于 Aoi 的内容更新算法 Main Function

1: 输入变量：U, F, T, C_s, C_b等系数
   计算 l_f, S,生成用户请求模型与m_{tf}
   初始化决策$(\boldsymbol{x_f}, \boldsymbol{a_f})$, k=0, stop=0
2: **while** stop = 0 **do**
3: 应用 CGA 求解最优策略，得到w_{fk}与$(\boldsymbol{x_f}, \boldsymbol{a_f})$
4: **if** w_{fk}为整数解
5: Stop = 1
6: **else**
7: 用 RA 来修改决策$(\boldsymbol{x_f}, \boldsymbol{a_f})$，$k$=$k$+1

8: **if** $k>N_0$

9: 采用舍入函数来收敛w_{fk}与$(\boldsymbol{x_f}, \boldsymbol{a_f})$

10: **end**

3.5 本章小结

本章主要介绍了缓存更新优化问题的求解策略。

首先，提出了基于重构的线性规划问题的最优策略求解方法，本文采用基于数学规划中的单纯形法产生的 CGA 的求解方法与 RA 联合应用来解决问题。对于 CGA，本文对其中涉及到的 RMP 问题与 SP 问题进行了数学表示。对于 SP 问题，提出了使用 Dijkstra 最短距离算法的求解方法，建立了从一个特定的 SP 问题到 Dijkstra 求解有向无环图的最短路径的映射关系。为了修改从 CGA 得到的分数解，设计了一个舍入算法，并添加了一个舍入函数使其能够快速收敛，最后总结了整个算法的求解流程。

第 4 章　仿真结果及分析

4.1　仿真结果与分析

在上一章中我们设计了用来得到最优更新策略的边缘缓存算法，为了展示并分析算法的性能，本章将构建的算法与已有的两种算法相比较。

最流行视频算法（Most-Popular-Video Algorithm，MPV），是一种基于流行度的基站缓存算法，目前被广泛应用于视频软件。MPV 是一种较主动的缓存策略，使用统计预测的视频流行度来提前缓存“最流行的视频”。它既不根据用户请求更新缓存，也不实施任何主动缓存替换策略，唯一引起缓存更新的因素是视频流行度分布的变化。由于能缓存的视频数量取决于缓存大小，因此针对互联网分布式服务的大型缓存实施 MPV 的缓存性能可能会很高，但是边缘缓存的大小有限，并且由于小用户请求的视频可能与最流行的视频有很大差异，因此用于边缘缓存的 MPV 策略实现的缓存性能可能不佳。此外 MPV 存在驱逐缓慢的问题：如果一个视频在短时间内变得非常流行，但很快就被遗忘了，那么该视频将需要很长时间才能被足够多的其他视频所取代，从而将其从缓存中逐出。

基于流行度的缓存更新算法（Popularity-based Algorithm，PBA）是一种基于 MPV 算法提出的改进算法[18]，相比于 MPV 算法，PBA 是一种主被动结合的缓存更新策略。一方面 PBA 根据用户请求来更新缓存，另一方面该算法采用统计预测的视频流行度来建立自己的“最不流行视频”组。此外 PBA 算法采用了一种添加基于已有视频流行度和最不流行视频组的缓存更新机制，从而避免了 MPV 驱逐缓慢的问题。该算法在大型缓存中性能略优于 MPV，但在小基站小缓存的系统中性能要远优于 MPV，然而两种算法都没有考虑信息年龄。

设立用户请求模型使得每个用户在每个时隙所请求的文件分布是可求的，进而得出每个文件的请求函数，即$h(u,r)$和$o(u,r)$，在仿真中设置每个用户在每个时隙发出一个请求。

本文设置参数 C_s=10，C_b=1，l_f为[1,10]的均匀分布，S为总文件大小的ρ倍，Zipf 分布系数γ=0.54，更新权重系数 λ=0.5，文件大小系数ρ=0.5，并且默认设置用户数$[U]$为 400，时隙数$|T|$为 10，文件数$|F|$为 100。在程序中一开始会生成用户请求模型，由于每次请求模型都具有随机性，在每一种参数设置下都重复运行 5 次三种算法，即每种参数设置下的文件请求数量至少为 400×10×5=20000 次，来尽可能消除随机性带来的误差。本文将用户请求通过三种算法得到的更新策略进行响应并计算总成本，系统总成本中是文件存储的成本，从服务器或基站向用户提供

文件的下载成本，文件更新成本，内容过时的惩罚成本等四种成本的和，其反映了系统的链路负载与缓存内文件的综合质量（新鲜程度与流行程度）。

表 4-1　算法参数

参数	意义
U	用户集
F	文件集
T	总时隙
C_s	从服务器下载一单元内容的花费
C_b	从缓存中下载一单元内容的花费
l_f	每个文件的大小(单元）
S	缓存大小
$p_f(i)$	内容储存成本
m_{tf}	请求内容f的用户数
λ	更新权重系数
ρ	缓存大小因子
k	迭代次数
γ	Zipf 分布系数

首先，本文仿真随着时隙增加三种算法对用户请求的响应情况，默认时隙数$|T|$为 10，在此项中，本文研究时隙数$|T|$从 10 到 20 下的三种算法的总系统成本，其余参数表示不变。

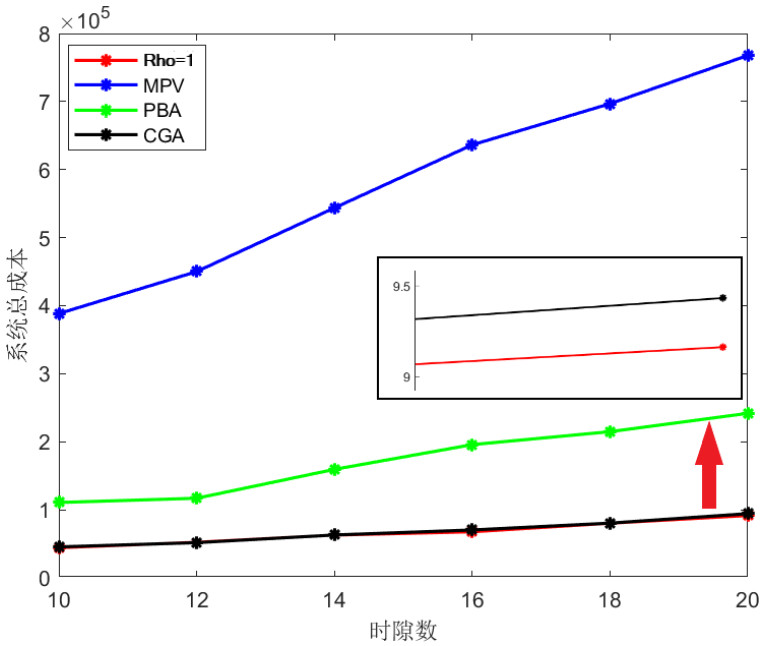


图 4-1　系统总成本随时隙数变化曲线

同时，为了研究设计的算法的性能与最优情况下的差距，在仿真时需要仿真最优情况。在该问题中，我设置该值为ρ=1 时的算法总成本。由式（2-3），边缘缓存大小与ρ和所有文件的大小之和有关，当ρ=1 时，缓存容量大到可以存储所有文件。即在此情况下，不存在从服务器直接提供某些内容给用户而不经过边缘缓存的过程。所有文件都将被存储在缓存中并按需更新，ρ=1 即代表缓存拥有无限容量。本文认为这种情况是该系统模型下边缘缓存的最优情况。

图 4-1 展示三种算法随着时隙的增加，系统总成本的变化情况。从图中可以看出，三种算法的总成本都随着时隙数$|T|$的增加而增加，本文设计的 CGA 算法在总成本上要相比另外两种算法有明显优势，且这种优势随着时隙数T的增加而变得更加明显。在只有 10 个时隙时，MPV 算法和 CGA 算法的总成本差异约为 3.5×10^5，而 PBA 算法与 CGA 算法的总成本差异约为 0.6×10^5，而当时隙数为 20 时，这两种成本差异就变为了6.9×10^5与 1.6×10^5。从图上可以明显看出，时隙每增长 1，MPV 算法和 PBA 算法的系统总成本增长要大于 CGA 算法，这意味着随着时隙数增多，即系统运行时间增长，CGA 算法的优势会越来越明显。由于时隙数增长可以模拟系统从短时间运行到长时间运行的情况，即图 4-1 显示了 CGA 算法在系统正常运行情况下的优势。

通过与理论情况下的总成本相比，本文设计的 CGA 提供的解决方案与缓存容量无限时的差距在 3%以内，并且相较另外两种算法 CGA 拥有更优的性能。一方面，CGA 算法的系统成本相比另外两者低，另一方面，随着时隙增长，CGA 算法的系统成本增长更缓慢。主要原因有两点：

其一是 MPV 算法和 PBA 算法在每个时刻做决策时都不考虑内容的新鲜度，如果有用户请求内容，系统就会将内容提供给用户，并且将基站中的内容更新，并没有考虑直接将基站中的内容发送给用户的这一操作。相比 CGA 算法而言，虽然采用另外两种算法的基站系统的缓存文件平均 AoI 较低，但实际上在每时隙的更新决策中做了许多不必要的下载策略，带来了相对高的系统负载。并不是所有用户请求的文件都需要极低的 AoI，比如一个混杂着视频与文档文件（如新闻、表格、论文等文件）的用户请求组：基站对于其中文档文件的请求处理就不需要每时隙都更新缓存中的对应内容，采用 MPV 和 PBA 的算法就会带来不必要的系统成本。

另一个原因是 CGA 算法的更新策略是考虑到了缓存容量S并经过了数学规划后的最优更新策略，而 MPV 和 PBA 算法的更新策略相比起来要更加被动，并且对于基站的有限容量S的考虑不够充分。即使是考虑到“最不流行视频组”的 PBA 算法也没有进行类似数学规划的求解过程。由于每一时隙更新前都进行了相应的规划来求解相对的策略来决定哪些文件要主动从缓存中删除，哪些文件需要更新，因

此时隙数|T|的增长对 CGA 算法的成本影响较小。

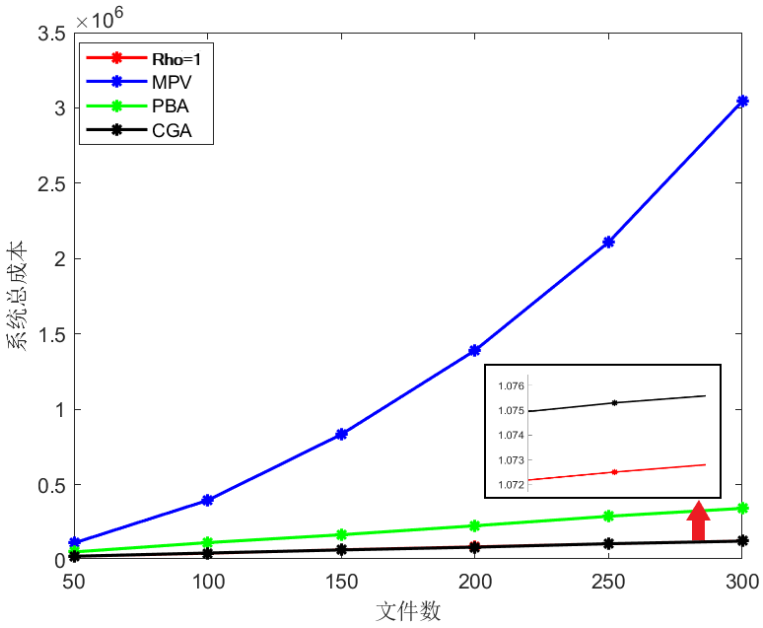


图 4-2　系统总成本随文件数变化曲线

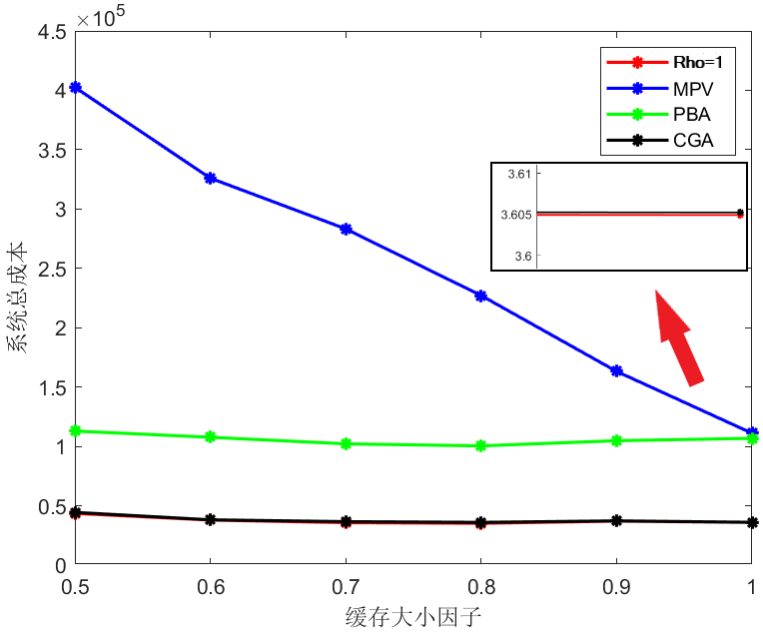


图 4-3　系统总成本随缓存容量变化曲线

图 4-2 为系统总成本随文件数变化的曲线。从|F|=50 到|F|=300，我们模拟了系统的服务器规模从小到大的过程，可以明显看出CGA算法和PBA算法相比MPV算法有巨大优势，且系统中文件数越多优势越明显，文件数越小三者的系统成本越接近。这是因为当|F|=50 时，系统缓存的容量非常有限，并且由于|F|很小，用户数远大于文件数，因此几乎所有的内容都会被请求。这些点意味着系统需要从服务器满足许多请求，基站作用较小，而三个算法的差异主要集中在基站更新策略上，

因此这种情况下三者成本接近。随着文件数增多，缓存容量同时也增加，而 CGA 能够有效地利用缓存容量，因此成本较低。

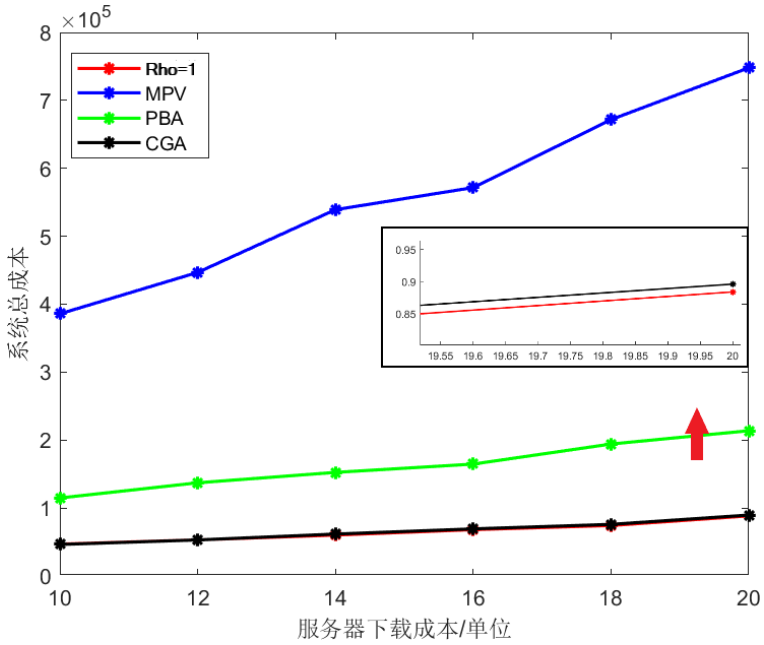


图 4-4　系统总成本随服务器下载成本变化曲线

图 4-3 为系统总成本随基站容量大小变化的曲线。基站缓存容量大小S为所有文件总大小的ρ倍，随着ρ从 0.5 增大到 1，缓存的容量逐渐增大到能存储所有文件。图中可以看出当ρ增大时，三种算法的性能差异在逐渐缩小，且当ρ=1 时 MPV 和 PBA 算法的性能几乎相同。PBA 是在基站的内容更换（删除部分内容与存储新内容）策略上基于 MPV 的改进算法，当ρ等于 1 时，基站可以存储所有文件，两种算法也就不存在性能差异。在该情况下，CGA 算法与最优情况的偏差来源于用户请求的随机性。而此情况下的 MPV 与 PBA 算法与 CGA 算法的性能差异是关于 AoI 的更新策略不同导致的。

图 4-4 为系统总成本随服务器下载成本变化的曲线，在此图中，改变了每单位大小的文件从服务器发送给用户的成本C_s，设置为从 10 到 20，而每单位大小的文件从基站发送给用户的成本C_b为 1 保持不变，如果假设基站离用户的距离不变，则两个成本的比值可以代表服务器-基站系统到用户的距离，即C_s越大，服务器离用户越远。从图中可以看出当服务器离用户近时，三种算法的性能差距较为接近，而随着C_s增大服务器离用户距离变远，三种算法的性能差距增大，MPV 算法的性能劣化最快。主要原因是三种算法对于基站更新策略的设计不同，MPV 算法在基站中只缓存那些被认为是最流行的内容，一旦某些内容的流行度在短时间内发生变化或者用户频繁请求那些流行度介于最流行和不流行之间的内容，该算法不会改变基站缓存中的内容，造成系统成本偏高。PBA 算法一定程度上避免了这个问

题，而本文设计的 CGA 算法比 PBA 算法考虑的更全面，因此系统成本更低。

上述仿真结果表明本文设计的CGA算法相较于现有的两种算法在系统成本上有明显优势，在文件数越多，工作时间越长，缓存容量越有限，服务器离用户越远时该算法的优势越明显。这是由于本文的算法在考虑到缓存容量的约束以及信息年龄AoI的基础上引入了一个数学规划的优化问题，所以其求解出的更新策略比单纯的依赖于用户请求的更新策略性能更优异。

然而由于本文算法引入了ILP以及Dijkstra最短路径算法，带来了求解时间长的问题。经过仿真观察，该算法在每一次迭代中主要于Dijkstra算法上花费时间。Dijkstra算法的时间复杂度[17]为$O(n^2)$，其中n为该算法使用的邻接矩阵的列数，n=$3+(3+T)\times\frac{T}{2}$，即n与$|T|$呈平方正比关系，时隙数$|T|$越大该算法所需计算时间越长。

4.2 本章小结

本章对算法进行了仿真测试与结果分析。通过数学软件仿真，本文将设计的算法与两种缓存更新算法进行了性能比较。本文从时隙数$|T|$，文件数$|F|$，基站缓存大小S和服务器下载成本C_s四个方面对设计的 CGA 算法与 MPV 算法和 PBA 算法来进行比较，结果表明四种情况下 CGA 算法在系统总成本上都有显著优势。从总体上看，越是文件多、时间长、离服务器远的小容量基站，也就是通常的边缘缓存系统采用的基站，CGA 算法的性能优势就越明显，采用该算法可以尽可能达成系统负载和缓存文件新鲜度的均衡。另外，本文分析了 CGA 算法相对另外两种算法拥有优势的可能原因，并且提出了该算法时间复杂度高的缺点。

结　论

1. 本文的主要内容

近年来全球总数据流量急剧增加，网络负载加重，为了对大量数据进行缓存，产生了提高缓存性能的边缘缓存技术，然而这项技术会使得缓存中存储的文件的新鲜度较低。目前已有的边缘缓存更新算法都着眼于信息的流行度，没有一种同时考虑信息的新鲜度和流行度的更新算法。

本文希望设计一种综合考虑以上两点的应用于有限缓存系统的更新算法，来减小系统的网络负载。首先通过对边缘缓存系统的考察，建立了在用户随机请求下的该系统内容存储的更新机制。为了找到在缓存容量有限的基站中考虑新鲜度和 AoI 的边缘缓存更新算法。我建立了在用户随机请求下的一个缓存容量有限的基站内容存储的数学模型并添加了一个基于 AoI 的惩罚成本，最终建立起一个优化问题。该问题是一个整数规划问题，通过数学方式，我证明了它是一个 NP-hard 难度问题并且可以通过数学规划的方法求解。

由于直接通过数学规划求解该问题，变量很多计算缓慢，本文对该问题做了重新表述并提出了一种基于单纯形法的迭代的解决方案，称为 CGA 算法。通过找到 CGA 算法的子问题与最短路径算法中有向无环图的映射关系，该算法在传统的求解整数规划问题的线性规划方法上结合了最短路径算法，减少了计算量，节约了大量的计算时间。

由于 CGA 算法得到的可能不是整数解，本文还添加了相关的舍入算法和函数，舍入算法用来使非整数解收敛到整数解，舍入函数是基于前期仿真写出的函数，用来提高整个算法的收敛速度。至此本文得到了一种综合考虑信息年龄与新鲜度的的有限容量边缘缓存的更新方案。

通过数学仿真软件，本文将设计的 CGA 算法与目前应用的的 MPV 算法和其改进算法，PBA 算法这三种算法进行了仿真并且与现有的更新方案进行比较，在时隙数$|T|$，文件数$|F|$，基站缓存大小S和服务器下载成本C_s四个参数变化的情况下进行仿真，并且研究它们对系统总成本的影响。结果表明四种情况下 CGA 算法在系统总成本上都有优势，且在多文件、长时间、离服务器远的小容量基站中采用 CGA 算法的性能显著优于另外两种算法。此外，仿真还显示 CGA 算法的性能接近缓存容量为无穷时的性能。本文通过分析该算法具有相对优势的可能原因，证明了在边缘缓存系统中采用设计的 CGA 算法可以达成系统网络负载和缓存文件新鲜度的均衡。

2. 今后工作的展望与设想

1）本文提出的边缘缓存模型考虑的用户请求模型较理想化，实际情况中用户请求的分布更随机，并不是每个时隙都会产生请求，这就要求在后续的研究中建立更实际的用户请求模型，考虑用户生成请求的数量和产生请求的时隙，进而提高模型的准确性。

2）如上文所述，相较于另外两种算法，该算法存在时间复杂度高，计算时间长的问题，究其原因是 Dijkstra 算法的性能限制，随着时隙数增长算法的运行时间呈几何级增长。在后续的研究工作中需要改善该问题，我认为该问题有两种方法，一种是限制时隙数，将时隙数$|T|$设置为不大于 10 的值，在基站上采用多次计算的方式避免单次计算时间太长的问题；另一种方法是找到性能更好的最短路径求解算法并建立起其与 SP 问题的映射关系，如 SPFA 算法。

参考文献

[1] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update [OL], 2017–2022 White Paper. Accessed: Feb. 2019

[2] D. Liu, B. Chen, C. Yang and A. F. Molisch, Caching at the wireless edge: design aspects, challenges, and future directions[J]. IEEE Communications Magazine, 2016, vol. 54, no. 9, pp. 22-28.

[3] Min Chen, Yixue Hao, Long Hu, Kaibin Huang, Vincent K. N. Lau. Green and Mobility-Aware Caching in 5G Networks[J]. IEEE Transactions on Wireless Communications, 2017, 16(12):8347-8361.

[4] P. Blasco and D. Gndz, “Learning-based optimization of cache content in a small cell base station,” in IEEE ICC, 2014, pp. 1897–1903.

[5] M. A. Kader, E. Bastug, M. Bennis, E. Zeydan, A. Karatepe, A. S.Er, and M. Debbah, “Leveraging big data analytics for cache-enabled wireless networks,” in IEEE GLOBECOM Wkshps, 2015, pp. 1–6.

[6] M. K. Kiskani and H. R. Sadjadpour, “Capacity of cellular networks with femtocache,” in IEEE INFOCOM Wkshps, 2016, pp. 9–14.

[7] B. N. Bharath, K. G. Nagananda, D. Gndz, and H. V. Poor, “Caching with time-varying popularity profiles: A learning-theoretic perspective,” IEEE Trans. Commun., vol. 66, no. 9, pp. 3837–3847, 2018.

[8] N. Zhang, K. Zheng, and M. Tao, “Using grouped linear prediction and accelerated reinforcement learning for online content caching,” in IEEE ICC, 2018, pp. 1–6.

[9] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal and N. B. Shroff, "Update or Wait: How to Keep Your Data Fresh," in IEEE Transactions on Information Theory, vol. 63, no. 11, pp. 7492-7508, Nov. 2017

[10] Tang, P. Ciblat, J. Wang, M. Wigger, and R. Yates, “Age of information aware cache updating with file-and age-dependent updatedurations,” https://arxiv.org/pdf/1909.05930.pdf, 2019.

[11] C. Kam, S. Kompella, G. D. Nguyen, J. E. Wieselthier, andA. Ephremides, “Information freshness and popularity in mobilecaching,” in IEEE ISIT, 2017, pp. 136–140.

[12] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, “Age-optimal con-strained cache updating,” in IEEE ISIT, 2017, pp. 141–145.

[13] M. S. Heydar Abad, E. Ozfatura, O. Ercetin, and D. Gndz, “Dynamiccontent updates in

heterogeneous wireless networks,” in IEEE/IFIP 15th WONS, 2019, pp. 107–110.

[14] G. Ahani and D. Yuan, "Accounting for Information Freshness in Scheduling of Content Caching," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1-6

[15] D. N. Serpanos, G. Karakostas and W. H. Wolf, "Effective caching of Web objects using Zipf's law," 2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532), 2000, pp. 727-730.

[16] M. R. Garey and D. S. Johnson, Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman, 1979.

[17] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma and N. Nosovic, "Dijkstra's shortest path algorithm serial and parallel execution performance analysis," 2012 Proceedings of the 35th International Convention MIPRO, 2012, pp. 1811-1815.

[18] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, “Femtocaching: Wireless content delivery through distributed caching helpers,” IEEE Trans. Inf. Theory, vol. 59, no. 12, 2013, pp.8402–8413

哈尔滨工业大学深圳校区本科生毕业设计（论文）原创性声明

本人郑重声明：在哈尔滨工业大学深圳校区攻读学士学位期间，所提交的毕业设计（论文）《基于信息新鲜度的有限容量边缘缓存更新算法设计》，是本人在导师指导下独立进行研究工作所取得的成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明，其它未注明部分不包含他人已发表或撰写过的研究成果，不存在购买、由他人代写、剽窃和伪造数据等作假行为。

本人愿为此声明承担法律责任。

作者签名：  日期：2022 年 5 月 24 日

致　谢

首先感谢罗晶晶老师为我提供必要指导。在她的悉心教导下我才得以完成这篇论文。从开题报告到结题论文，罗老师给出了许多宝贵的意见和建议，尤其是在格式方面对我的论文提出了高要求，帮我纠正了许多格式上的小错误。我还要感谢琬璐学姐对我的帮助，论文中的某些算法受到了她的启发，并且琬璐学姐还帮助我完善了自己的程序，让我对于报告的撰写有了更清晰的思路，她也对我的论文撰写提供了非常多的建议。

此外，感谢我的朋友们，现在我觉得出国留学就是一场与孤独搏斗的过程。北欧的冬天每天光照不到五个小时，漫长的黑夜给了我巨大的心理压力，再加上刚接触该项目时理解不透彻，进度缓慢，从去年的 11 月到今年 4 月入春期间精神衰弱近似抑郁，一度以为自己会无法完成毕业设计。在朋友们的鼓励下我才得以坚持下来。

最后，我要感谢评审的老师对我的论文进行评阅指正。希望疫情早日结束，祝祖国永远繁荣富强。