arXiv:2301.07007v1 [cs.DS] 17 Jan 2023

图流中的全范数负载均衡：基于乘法权重更新方法

Sepehr Assadi AaronBernstein Zachary Langley

2023年1月18日

摘要

在加权负载均衡问题中，输入是一个包含 n 个顶点的二分图，其中一侧是客户集合，另一侧是服务器集合，且每个客户都带有一个非负实数权重。输出是一种分配，将每个客户映射到与其相邻的某台服务器；于是，服务器的负载就是分配给它的客户权重之和。目标是找到一种均衡良好的分配，通常通过（近似地）最小化服务器负载的 ℓ_∞-范数或 ℓ_2-范数来衡量。全范数负载均衡问题将这两个目标加以推广，要求找到一种分配，同时近似最小化适用于 $p \geq 1$ 的 *all* ℓ_p-范数目标，包括 $p = \infty$。

我们的主要结果是一种用于全范数负载均衡问题的确定性半流式算法，它使用 $O(\log n)$ 趟，并提供 $O(1)$-近似。在我们的工作之前，在半流式设置中，已知的只有一种针对 ℓ_∞-范数目标的算法，它使用 $O(\log n)$ 趟，并提供 $O(\log n)$-近似。

我们的算法将乘法权重更新方法创新地应用于一个混合覆盖/打包凸规划，以解决包含无限多个约束的全范数负载均衡问题。

1 引言

在负载均衡问题中，输入是一个连接客户C和服务器 S的二分图 $G=(C,S,E)$，以及一个客户端权重函数 $w: C \to \mathbb{Z}^+$。目标是找到一种尽可能“均衡”的分配——将每个客户映射到其相邻服务器之一。给定一种分配时，某台服务器的负载是分配给它的客户权重之和；负载向量则是按服务器索引排列的服务器负载向量。“均衡”这一概念可以通过负载向量的适当范数来刻画：范数越小，分配就越均衡。例如，常见的“最小化最大值目标”——即最小化最大服务器负载——可以通过最小化 ℓ_∞-范数来实现。尽管最小化最大值目标很自然，但它并非总是合适，因为它可能无法妥善均衡那些不处于主要瓶颈中的客户。因此，另一个常见目标是最小化 ℓ_2-范数。

同时推广 ℓ_∞-范数和 ℓ_2-范数目标，可以考虑寻找一种分配，使其负载向量的 ℓ_p-范数对于所有 $p \geq 1$均达到最小，包括 $p=\infty$，并且同时满足这些目标。一般而言，这样的分配可能不存在，[2], 但或许令人惊讶的是，总是存在一种分配，对于每个 ℓ_p-范数同时都是 2-近似 [10]（另见 [29,15, 18]）。同时计算一个对所有ℓ_p-范数均近似最优的分配的问题，被称为全范数负载均衡问题。

我们在半流式模型 [23] 中研究全范数负载均衡问题：一个 n 个顶点的图的边以数据流形式逐条到达，算法可以使用 $O(n\text{polylog } n)$ 个字的内存处理这些边，并且只需对数据流进行少量遍。这样的约束体现了处理海量图时面临的多项挑战，包括 I/O 效率以及对不断演化的图进行高效监测。因此，近年来半流式模型受到了广泛关注。

负载均衡问题在不同模型下以不同术语得到广泛研究。在调度文献中，它有着悠久的历史，被称为受限分配作业调度[30, 17, 39, 29]。在分布式计算文献中，它被作为备份放置问题 [28, 43, 11, 6] 进行研究（另见 [44, 40, 36, 46, 47] ，其中讨论了类似模型）。负载均衡也以半匹配 [29, 21,22, 35], 这一术语进行研究；从优化角度看，它是混合装箱-覆盖优化问题的典型例子之一。

负载均衡可视为最大匹配问题与覆盖问题的交汇；可以说，这两类问题是半流式模型中研究最多的优化问题（匹配问题的相关研究及其参考文献见 [24, 41, 1, 26, 34, 32, 20, 33, 8, 25, 7, 13, 5, 14] ，覆盖问题的相关研究见 [4, 12, 37, 42] ——这绝不是对已有成果的全面罗列）。然而，令人意外的是，人们对该模型中的负载均衡问题知之甚少。据我们所知，此前唯一研究半流式模型中负载均衡问题的工作由 Konrad 和 Ros´en [35], 完成，他们提出了一个单遍 $O(\sqrt{n})$-近似算法，以及一个需要 $O(\log n)$ 遍的 $O(\log n)$-近似算法；这两种算法都仅适用于最小化最大值目标和无权客户端。[1] 本文旨在弥补这一不足。

[1]最近的一项工作[16] 也研究了半流式模型中的混合装箱-覆盖线性规划。当这篇论文并未明确讨论负载均衡问题时，其结果似乎意味着，在 O最小化最大值目标下，对于（无权）问题可达到 (1)-近似，并且需要 $O(\log n)$ 遍 [45]。我们将在本文稍后讨论这一联系。

1.1 我们的结果

我们的主要结果如下定理所述。

定理 1.1 （形式化表述见 定理4.24）。 对于带加权客户的全范数负载均衡问题，存在一种确定性半流式算法，可实现$O(1)$-近似[2] ，使用 $O(\log n)$ 遍数据流。

定理 1.1 将 [35] 的半流式算法大幅推广到了加权客户和全范数保证。同时，该定理将近似比从 $O(\log n)$ 提高到 $O(1)$。

值得指出的是，即使对于无权最小最大目标，要获得优于 2-近似的结果，其难度至少不低于在输入图中寻找完美匹配。事实上，当且仅当图中存在客户端完美匹配时，最小最大目标的值为 1；否则，该值至少为 2。在半流式模型中，计算完美匹配至少需要 $\Omega(\log n/\log\log n)$ 遍 [27, 9, 19],，而目前最好的上界是 $n^{3/4+o(1)}$ 遍 [7]。

对于加权客户，35年来，寻找一种能以多项式时间为最小化最大值目标实现优于 2-近似的算法一直是一个公开问题，而要实现优于 1.5-近似则是 NP难的[38]。事实上，任何像我们这样使用 [38], 的算法对分数分配进行舍入，都无法达到优于 2-近似的效果，即使仅针对 ℓ_∞范数也是如此。因此，要将我们的算法在定理 1.1中的近似比改进到低于 2，似乎超出现有技术的能力范围。

1.2 我们的技术

我们通过使用乘法权重更新（MWU）方法，近似求解全范数负载均衡的凸规划，从而得到定理 1.1中的算法。乘法权重更新方法是流式模型中一种尤为成功的优化技术[1, 16, 31] ，因为它提供了一种方法，可以将“平均而言可行”的解提升为可行且近似最优的解。要应用乘法权重更新方法，需要设计合适的预言机，并证明该预言机能够通过半流式算法高效实现（参见第 2 节）。

负载均衡的最小-最大目标可以表述为一个混合打包-覆盖线性规划（LP）。最近的工作 [16] 提出了利用基于乘法权重更新的流式技术求解流式处理中的混合打包-覆盖线性规划，从而可以用来解决这一问题。然而，全范数负载均衡更具挑战性；它包含无穷多个非线性约束，因此，之前的基于乘法权重更新的流式技术 [1, 16] 并不适用。

从总体上看，我们将凸规划中用于全范数负载均衡的所有无穷多个装箱约束交由 MWU 预言机处理，并且只使用 MWU 本身来满足（线性的）覆盖约束。粗略地说，这要求 MWU 的预言机在每次迭代中，找到一种分配方式，将很大一部分（但关键是并非全部）客户分配给服务器，并且这种分配同时满足每一种ℓ_p范数下的负载均衡。实现这一预言机是我们工作的主要技术要素。我们将这项实现的详细内容留到第3 小节的精简方法概述中介绍。

[2]更准确地说，我们证明了我们的算法可以达到 19-近似。我们没有太多尝试去优化这个常数。近似因子似乎可以降至 8，但如果不进行实质性修改，则无法低于 4。

2 预备知识

我们用 $\mathbb{R}_+$ 表示非负实数。若 $f : \Omega \to R$ 是底集 Ω 上的某个函数，我们通过定义 $f(A) = \sum_{a \in A} f(a)$，将 f 扩展到 Ω 的子集 $A \subseteq$ 。

在全文中，$G = (C, S, E)$ 表示一个由客户 C 和服务器 S 构成的二分图，$w : C \to \mathbb{R}_+$ 是定义在客户上的权重函数。我们用 cs 表示边 $(c, s) \in E$ 。对于顶点 v，我们用 $\delta(v) = \{e \in E : e \ni v\}$ 表示与 v 相 incident 的边集。

对于函数 $x : E \to \mathbb{R}$，我们定义 x^+ by $x^+(e) =$ 最大值$\{x(e), 0\}$。

分配 部分分配 $A \subseteq E$ 是一个边集，其中每个客户在 A 中的度至多为一。若客户 c 被分配给 s （由 A），即 $cs \in A$，我们就称其被分配。对于 $s \in S$，我们用 $A^{-1}(s)$ 表示由 A 分配给 s 的客户集。若 A 中每个客户的度恰好为一，我们就简称 A 为分配。每个分配也是部分分配。部分分配 A 的负载函数 $L_A : S \to \mathbb{R}_+$ 定义为$L_A(s) = \sum_{c \in A^{-1}(s)} w(c)$；对于 $s \in S$，我们称 $L_A(s)$ 为 s 的负载（在 A 下）。部分分配 A 的 ℓ_p-范数为

$$\left(\sum_{s \in S} L_A(s)^p \right)^{1/p}.$$

函数 $z : E \to \mathbb{R}_+$ 是部分分数分配，当且仅当对于所有 $c \in C$，$z(\delta(c)) \leq 1$ 。如果对于所有 $c \in C$，$z(\delta(c)) = 1$ ，那么我们称 z 为分数分配。部分分数分配 z 的负载函数 L_z 定义为 $L_z(s) = \sum_{s \in S} \sum_{c \in N(s)} w(c) z(cs)$。分数分配 A 的 ℓ_p-范数为

$$\left(\sum_{s \in S} L_z(s)^p \right)^{1/p}.$$

b-匹配。对于任意函数 $b : V \to \mathbb{R}_+$，分数 b-匹配是满足以下条件的函数 $x : E \to \mathbb{R}_+$ ：对于所有 $v \in V$，$x(\delta(v)) \leq b(v)$。价值 $b(v)$称为 v的容量。若 $x(\delta(v)) = b(v)$，则称顶点 v 被 x 饱和。分数 b-匹配 x的大小为 $x(E) = \sum_{e \in E} x(e)$。如果对于每个分数 b-匹配 y，都有 $\alpha x(E) \geq y(E)$，则称 x 为 α-近似分数 b-匹配。如果对于所有 $e \in E$都有 $x(e) \geq y(e)$，则称 x 包含一个 b-匹配 y ，我们也将其记为 $x \geq y$。

在处理由客户和服务器构成的二分图时，分别指定客户和服务器的顶点容量往往会很方便。为此，对于$\kappa : C \to \mathbb{R}_+$和 $\tau : S \to \mathbb{R}_+$，分数$(\kappa, \tau)$-匹配是满足以下条件的分数 b-匹配

$$b(v) = \begin{cases} \kappa(v) & v \in C \\ \tau(v) & v \in S \end{cases}.$$

全范数负载均衡 在全范数负载均衡问题中，输入是一个二部图 $G = (C, S, E)$，以及一个权重函数 $w : C \to \mathbb{Z}_+$。我们将 C 中的元素称为客户，将 S 中的元素称为服务器。目标是找到一个分配 $A \subseteq E$ ，使其对于每一种 ℓ_p范数目标都同时近似最优。事实上，我们只能求近似解，因为不可能总是同时最小化所有 ℓ_p范数。但总存在一个分配，其 ℓ_p范数至多是最优 ℓ_p范数的两倍，对所有$p \geq 1$ [10]均成立。

如果对于所有的 $p \geq 1$，都有 $\|L_A\|_p \leq \beta \cdot \mathrm{OPT}_p$，则称分配 A 为 β-近似全范数指派，或更简单地称为 β-全范数指派，其中 OPT_p 是使 ℓ_p范数最小化的分配的 ℓ_p范数。

乘法权重更新方法 乘法权重更新（MWU）方法是一种通用且强大的元算法，除了其他用途外，还可用于在由半空间部分定义的凸集中寻找（近似）可行点。

假设我们有一组由 $a_i \cdot z \geq b_i$ 给出的、包含 n 个关于 m 个变量 $\mathbf{z} \in \mathbb{R}^m$ 的线性不等式，其中 $a_i \in \mathbb{R}^m$ 且 $b_i \in \mathbb{R}$ 对于 $i \in \{1, \ldots, n\}$。进一步假设存在一个凸集 $\mathcal{P} \subseteq \mathbb{R}^m$，我们希望找到一个 $z \in \mathcal{P}$，使得 z 满足上述线性不等式。我们将线性不等式 $a_i \cdot z \geq b_i$ 称为“硬”约束，将构成凸集的约束 $\mathcal{P}$ 称为“软”约束。乘法权重更新方法为每个约束 $i \in [m]$ 引入一个价值 λ_i，并将任务简化为：找到一个 $z \in \mathcal{P}$，使得 $\sum_{i=1}^m \lambda_i a_i \cdot z \geq \sum_{i=1}^m \lambda_i b_i$。为乘法权重更新方法服务、用于解决这一简化任务的算法称为预言机。

并非所有预言机都一样；尽管乘法权重更新（MWU）方法可以使用任意预言机来求解原问题，但能够生成“更接近”满足实际约束的解的预言机，所需的调用次数更少。

定义 2.1. 一个 d-有界预言机是一种算法，给定向量 $\lambda \in \mathbb{R}_+^m$，计算一个 $z \in \mathcal{P}$，使得

$$\sum_{i=1}^m \lambda_i a_i \cdot z \geq \sum_{i=1}^m \lambda_i b_i$$

或者正确报告不存在这样的 z。此外，对于所有 $i \in [m]$，都有 $|a_i z - b_i| \leq d$。

算法 1：乘法权重更新（MWU）方法。

给定：一个用于可行性问题的 d-有界预言机，以及一个误差参数 $\varepsilon \in (0, 1)$。

1. 设置 $\eta \leftarrow \varepsilon/(4d)$。设置 $T \leftarrow 8d^2 \log n/\varepsilon^2$。对于每个约束，设置 $\lambda_i^{(1)} \leftarrow 1 \quad i \in [n]$。
2. 对于 $t = 1$ 到 T，
3. 运行给定的 d-有界预言机，并使用 $\lambda^{(t)}$ 得到解 $z^{(t)}$。如果预言机返回不可行，则返回不可行。
4. 设置 $m_i^{(t)} \leftarrow$ 对于每个 $i \in [n]$，设置 $\frac{1}{d}(b_i - A_i z^{(t)})$。
5. 为每个 $i \in [n]$ 设置 $\lambda_i^{(t+1)} \leftarrow \lambda_i^{(t)}(1 - \eta m_i^{(t)})$。
6. 返回 $\frac{1}{T}\sum_{t=1}^T z^{(t)}$

下述定理由此得出 [3]，并且根据算法 1 若获得该设置下的预言机，则可在流式设置中实现。

定理 2.1。设 $\varepsilon > 0$ 是一个误差参数。假设我们有一个定义约束 $a_i z \geq b_i$的数据流。此外，假设存在一个用于可行性问题的 d-有界预言机，它对数据流进行 p 遍处理，并使用 s 空间。那么，配备该预言机的 算法 1 要么返回一个 x，使得 $a_i \cdot x \geq b_i - \varepsilon$，要么正确判定该系统不可行。该算法对数据流进行 $O(pd^2 \log m/\varepsilon^2)$ 遍处理，并使用$O(n + sd^2 \log m/\varepsilon^2)$ 空间。

3 高级概述

我们的目标是在半流式设置下计算一个 $O(1)$-全范数指派，并对数据流进行$O(\log n)$ 遍处理。我们的方法可以视为以下一系列归约:

1. 计算 $O(1)$-全范数分配可归约为计算 $O(1)$-全范数分数分配。2. 计算 $O(1)$-全范数分数分配可通过乘法权重更新方法，归约为求解我们称为 $O(1)$-全范数预言机问题的问题。3. $O(1)$-全范数预言机问题又可归约为计算规模因子为 $O(1)$ 的匹配层级。4. 最后，计算规模因子为 $O(1)$ 的匹配层级可归约为计算 $O(\log^2 n)$ 个极大 b-匹配。

第一步所用的归约算法广为人知，归功于 Lenstra、Shmoys 和 Tardos [38]。第二步的归约采用乘法权重框架，应用于刻画 $O(1)$-全范数分数分配问题的无限约束凸规划。第三步和第四步是我们的主要算法贡献。大部分复杂性集中在第三步，本节将阐释这一步骤背后的直觉。

$O(1)$-全范数预言机问题。在 β-全范数预言机问题中，输入为一个图 $G=(C,S,E)$、权重 $w\colon C\to\mathbb{R}_+$以及值 $\lambda\colon C\to\mathbb{R}_+$。目标是计算一个函数 $z\colon E\to\mathbb{R}_+$，使得

$$\sum_{c\in C}\lambda(c)z(\delta(c))\geq\sum_{c\in C}\lambda(c)$$

and

$$\left(\sum_{s\in S}\Big(\sum_{c\in N(s)}w(c)z(cs)\Big)^p\right)^{1/p}\leq\beta\cdot\mathrm{OPT}_p$$

对于所有 $p\geq 1$，其中 OPT_p 是使 ℓ_p范数最小化的分配的 ℓ_p范数。若某个算法能够解决某个常数 β对应的 β-全范数预言机问题，我们就称该算法解决了 $O(1)$-全范数预言机问题。

不妨假设对所有 $c\in C$，$w(c)=1$。这一假设并非无损一般性，但由于它已涵盖我们方法中的许多关键算法思想，为简化下面的讨论，我们采用这一假设。

只需找到一个部分分配 $A\subseteq E$，使得

$$\sum_{cs\in A}\lambda(c)\geq\frac{1}{\alpha}\sum_{c\in C}\lambda(c).\tag{1}$$

对于某个 $\alpha\geq 1$，并且

$$\left(\sum_{s\in S}|L^{-1}(a)|^p\right)^{1/p}\leq O(1)\cdot\mathrm{OPT}_p.\tag{2}$$

事实上，将 A 的指示函数缩放 α 后即可得到一个解。

部分分配的收费方案。设 $A \subseteq E$ 为一个部分分配，$\prec$ 为 C的任意全序。现在，我们描述一种有用的收费方案，它根据 $\prec$将数量 $\|L_A\|_p^p$ 分配给客户。更正式地说，对于每个满足以下性质的 $p \geq 1$，我们定义一个函数 $\phi_p \colon C \to \mathbb{R}_+$

$$\sum_{c \in C} \phi_p(c) = \|L_A\|_p^p. \tag{3}$$

定义很简单：如果 A 不分配 c，则设置 $\phi_p(c) = 0$。否则，如果 c 是按 $\prec$度量时，在分配给 $A(c)$ 的客户中第k小的元素，则设置$\phi_p(c) = k^p - (k-1)^p$。容易看出，由于求和项会依次抵消，$\sum_{c \in A^{-1}(s)} \phi_p(c) = L_A(s)^p$。因此，$\phi_p$ 满足所需的条件 (3)。

该收费方案将作为上界的依据，用于界定我们的算法计算出的解的成本；同时也将作为最优成本下界的依据。

匹配层次结构与 ℓ_p-范数。为了解决 $O(1)$-全范数预言机问题——即找到满足 (1) 和 (2) 的 $A \subseteq E$ ——我们首先计算一个匹配层次结构。令$\ell = \lceil \log_2 n \rceil$。匹配层次结构是匹配 $M_0, \ldots, M_\ell$ 构成的序列，使得 M_i 是一个 $(1, 2^i)$-匹配，并且 $M_0 \subseteq \cdots \subseteq M_\ell$。根据定义，如果客户 c 在某个 M_i中与服务器 s匹配，那么对于所有 $i' \geq i$，它在 $M_{i'}$ 中都与 s 匹配。令客户 c 的秩为满足以下条件的最小 i：c 在 M_i 中匹配；如果不存在这样的 i，则为 ∞。

匹配层级有两大优点。首先，它为构建部分分配提供了基础。注意，每个 M_i 都是最大负载为 2^i的部分分配；由于这些 M_i彼此嵌套，每个将客户 c 分配给服务器的 M_i，对将 c 分配到哪台服务器的判断都一致。最终，我们的解 A 将是 M_ℓ的子集；也就是说，我们将依据 M_ℓ 分配部分客户，而让其他客户保持未分配状态。

匹配层级的第二个优点，也是它区别于简单分配之处，在于它能自然地为客户定义一种有用的排序；任何按秩非递减顺序排列客户的方式都可以。要了解这种排序为何有用，请考虑一个 $A \subseteq M_\ell$。如果 c 由 A 分配，且 c 的秩为 i （在匹配层级中），那么 c 必须出现在 A中分配给 s 的前 2^i 个客户之列。因此，$\phi(c) \leq 2^{ip} - (2^i - 1)^p$. 由匹配层级定义的客户秩所构成的函数，为 $A \subseteq M_\ell$ 的 ℓ_p-范数提供了一个上界。

我们还可以用匹配层级给出 OPT_p^p 的一个下界。令 A^* 为一个使 ℓ_p-范数最小化的分配，并考虑客户的任意一种排序。对于 $i \in \{0, \ldots, \ell\}$，定义 M_i^* 为将每台服务器分配给其最先的 2^i 个客户的 $(1, 2^i)$- 匹配。现在考虑与之前相同的收费方案，只是这次将其应用于 M_ℓ^*。根据 M_0^* 的定义，秩为 0 的客户 c 是其在 A 下的服务器原像中的第一个元素，因此 $\phi(c) = 1$。秩为 i 的客户 c 在其于 A 下的服务器原像中至少有 2^{i-1} 个客户排在前面，因此 $\phi(c) \geq (2^{i-1} + 1)^p - 2^{(i-1)p}$。

主要结论如下。我们计算得到的匹配层级 A 为每个客户赋予一个秩；我们也可以将一个最优解分解为另一个匹配层级，并由此为每个客户定义其自身的秩。为了证明 A 接近最优，我们比较 A 中客户的秩与最优解中客户的秩。

规模因子。仅仅拥有许多秩较低的客户还不够；我们还需要确保分配给客户的 λ-价值足够高，否则我们可能无法满足（1）。一种

方法是确保每个 M_i 匹配到的 λ–价值，是任何其他 $(1, 2^i)$-匹配所匹配到的 λ-价值的一个常数比例。我们希望得到稍强一些的结论。

设 $C^{(j)} = \{c\colon \lambda(c) \geq \max_{c \in C} \lambda(c) \cdot 2^{-j}\}$，设 $E^{(j)} = \{cs \in E\colon c \in C^{(j)}\}$，并设 $G^{(j)} = (C, S, E^{(j)})$。我们称匹配层级 $M_0, \ldots, M_\ell$ 具有规模因子 α，如果对于所有 $i, j \in \{0, \ldots, \log n\}$，$M_i \cap E^{(j)}$ 都是 $G^{(j)}$ 的一个 α- 近似最大基数 $(1, 2^i)$- 匹配。

注意，具有规模因子 α 尤其意味着，每个M_i 匹配到的价值至少是任何其他 $(1, 2^i)$-匹配所匹配价值的 $1/\alpha$ 。也就是说，M_i 是一个 α-近似最大“权重”匹配，其中边 cs 的“权重”定义为 $\lambda(c)$（加引号是为了强调，这种“权重”不同于客户 c的权重 $w(c)$）。

要理解这种匹配层级为何有用，请看下面的论证。我们要证明，与最优匹配层级相比，我们的解中有许多高成本的低秩客户。任取一个秩阈值 i，并注意到，$G^{(j)}$ 中最大的 $(1, 2^i)$-匹配恰好等于在 A^* 中秩为 $\leq i$ 且价值为 $\geq \max_{c \in C} \lambda(c) 2^{-j}$ 的客户数量。因此，规模因子为 α 的匹配层级中，这类客户的数量至少是 $1/\alpha$ 的 A^* 倍。换言之，对于任意秩/成本权衡，该匹配层级都能与 M_0^*、.....、 M_ℓ^* 相媲美。

行不通的方法。计算出匹配层级 $M_0, \ldots, M_\ell$ 后，我们需要计算一个满足 (1) 和 (2) 的部分分配。一种简单的做法是直接令部分分配为 M_ℓ 。

要了解这种做法为何行不通，请看下面的例子。设 n 是 2 的幂，设 $C = \{c_1, \ldots, c_n\}$，并设 $S = \{s_1, \ldots, s_n\}$。图 G 是由 C 和 S 构成的完全图，并且对所有 $c \in C$，$\lambda(c) = 1$ 。嵌套匹配层级如下。匹配 M_0 将 c_k 与 s_k 匹配，其中 $k \in \{1, \ldots, n/2\}$。对于 $i \geq 1$，每个 M_i 都通过将先前尚未分配的客户分配给 s_1，在 M_{i-1} 的基础上构建。由于每个 M_i 都匹配前 $n/2$ 个客户，该匹配层级的规模因子显然为 2。但是， s_1 在 M_ℓ 中的负载为 $n/2$ ，而最优分配的 ℓ_∞-范数为 1（G 存在完美匹配）。

因此，我们的想法不是选取所有 M_ℓ，而是让 A 根据 M_ℓ 为一部分严格子集的客户分配任务，并让其余客户保持未分配状态。结合上面的例子，另一个看似诱人的方案是按秩递增的顺序选取客户，直到收集到足够的 λ-价值。然而，当低秩且价值较低的客户很多时，这种方法就会失效。

上一段所述方法的问题在于，它没有考虑客户的价值。同样短视的方法是忽略秩，按价值递减的顺序分配客户，直到满足（1）。我们的第一个例子已经说明了这种方法为何行不通。

因此，在挑选客户子集时，我们需要在客户的成本与价值之间保持平衡。对于 ℓ_p-范数，我们已经有一个很好的客户成本近似指标：如果设置 $\hat{\phi}_p(c) = 2^{ip} - (2^i - 1)^p$ ，其中 i 是 c的秩，那么 $\hat{\phi}_p(c) \leq \phi_p(c)$。因此，我们可以为每个客户分配成本和价值，而这个问题就类似于分数背包问题。因此，自然可以尝试采用类似分数背包问题的 $\hat{}$ 算法：按照 $\phi_p(c)/\lambda(c)$递减的顺序，贪心地分配客户，直到满足条件 (1)。事实上，如果只关注最小化某个固定的 ℓ_p-范数，这种方法是可行的。全范数负载均衡的问题在于， ϕ_p 是 p的函数，而它所导出的排序取决于 p的选择。

我们的算法。我们的算法按以下方式优先考虑价值。首先，它只考虑 $C^{(0)}$中的客户。算法从 $C^{(0)}$ 开始，按照秩递增的顺序收集客户。接下来这一步至关重要。一旦算法收集到恰好 $\frac{1}{\alpha}|C^{(0)}|$ 个客户，[3] 就结束这次 $C^{(0)}$ 迭代。随后，算法将纳入考虑的客户集合扩展至 $C^{(1)}$，并继续按秩递增的顺序添加尚未选取的客户。再次收集到恰好 $\frac{1}{\alpha}|C^{(1)}|$ 个客户后，它会继续处理 $C^{(2)}$。以此类推。

不难看出，该算法至少能收集到 $\lambda(C)$ 的 $1/\alpha$ 比例。具体来说，先从 $C^{(0)}$ 收集 $|C^{(0)}|/\alpha$ 个客户，再从 $C^{(1)}$ 收集 $C^{(1)}/\alpha$ 个客户，依此类推，最终该算法将获得总价值的 $1/\alpha$ 比例。

匹配层次结构为我们提供了一个关键工具，用于证明某个分配对于所有 ℓ_p-范数而言都是近似最优的。对于每个 i, j，令 $N_{i,j}$ 表示我们的秩为 $\leq i$、价值为 $\geq \max_{c\in C}\lambda(c)2^{-j}$ 的层次结构中的客户数量，并令 $N^*_{i,j}$ 表示最优匹配层级中对应的客户数量。如果我们能够保证对于所有 i, j，都有 $N_{i,j} \sim N^*_{i,j}$，那么不难验证，我们的解对于所有范数而言都是近似最优的（考虑上文所述的收费方案即可）。我们并未完全达到 $N_{ij} \sim N^*_{ij}$，但实现了一个略弱的保证，而这仍足以确保对于所有范数的近似最优性。

加权客户。从加权分数负载均衡到无权负载均衡，有一种众所周知但效率低下的归约：将每个客户 c 替换为 $w(c)$ 个单位权重的副本，并为每个客户赋予价值 $\lambda(c)/w(c)$。事实上，客户复制图中的任何分配都可以映射回原图中的分数分配，且负载向量相同。由于加权图中的每个分配也对应于客户复制图中的一个分配，并且二者的 ℓ_p-范数相同，因此，客户复制图中的一个 β-近似全范数分配，就是原图中的一个 β-近似全范数分数分配。

我们没有显式地通过这一归约运行流式算法，因为至少在直接应用时，它在半流式设置下行不通；相反，我们直接维护一个分数分配。不过，所有直觉都源自客户复制这一思路。例如，在无权算法中，我们按价值递减的顺序考虑客户。在加权算法中，为了模拟客户复制图上的无权算法，我们按 $\lambda(c)/w(c)$ 递减的顺序考虑客户。

4 流式模型中的全范数负载均衡

4.1 All-NormOracle 问题

在本小节中，我们将展示，乘法权重框架结合经典舍入算法，如何将 $O(1)$-全范数负载均衡任务归约为一个更简单问题的求解，我们称之为 $O(1)$-全范数预言机问题。建立归约之后，读者便可以不再关注乘法权重更新方法，而专注于在半流式设置中求解 $O(1)$-全范数预言机问题。

Rou分配。要计算一个稀疏的 $O(1)$-全范数分配，只需计算一个 sp
稀疏的 $O(1)$-全范数分数分配。事实上，Lenstr 的经典算法 a,

[3]为简化起见，我们假设 α 整除 $|C^{(j)}|$。我们的实际算法计算的是一个分数部分分配，因此不存在整除性问题。

Shmoys 和 Tardos [38] 就能解决问题。

引理 4.1（[10]）。 *[38]* 的舍入算法以分数分配$z\colon E \to \mathbb{R}_+$为输入，并生成一个（整数）分配 $A \subseteq E$，使得 $\|L_A\|_p \le \|L_z\|_p + \|w\|_p$. 此外，该算法使用 $O(\text{supp}(z))$ 空间。

因此，如果我们能在半流式设置中计算出一个 $O(1)$-全范数分数分配，且其支撑集大小为 $O(n\text{polylog } n)$，那么只需使用引理 4.1将其作为后处理步骤进行舍入即可。

$O(1)$-全范数预言机问题。为了在半流式设置中计算一个 $O(1)$-全范数分数分配，我们采用乘法权重更新框架。回顾一下， β-近似全范数分数分配 z 必须满足两类约束。首先是客户端覆盖约束，即线性约束 $\sum_{s\in N(c)} z(cs) \ge 1$ ，对每个 $c \in C$ 均成立。此外还有 β-近似的ℓ_p-范数约束，即凸约束

$$\left(\sum_{s\in S}\left(\sum_{c\in N(s)} w(c)z(cs)\right)^p\right)^{1/p} \le \beta \cdot \mathrm{OPT}_p,$$

对于所有 $p \ge 1$。

为了应用乘法权重更新方法，我们将 β-近似 ℓ_p-范数约束视为定义凸集 $\mathcal{P}$ 的条件。虽然此类约束有无穷多个，但可行点构成一个凸集，因此约束的不可数性不会给该框架带来问题。

在这一表述中，乘法权重更新方法为每个客户 c 引入一个价值 $\lambda(c)$。接下来要设计一个预言机，计算一个 $z\colon E \to \mathbb{R}_+$，使得对于所有 $p \ge 1$， β-近似ℓ_p-范数约束均得到满足，并且

$$\sum_{c\in C} \lambda(c) z(\delta(c)) \ge \sum_{c\in C} \lambda(c). \tag{4}$$

我们将约束（4）称为组合客户端覆盖约束。该问题的正式表述见图 1。

W我们现在可以证明本节的主要引理
。

引理 4.2. 如果存在一个针对全范数预言机问题的 d-有界、 p-遍、 $S(n,m)$-空间流式算法，那么就存在一个 $O(pd^2\beta^2 \log n)$-遍、 $O(S(n,m)\cdot \beta^2 d^2 \log n + n)$-空间流式算法，用于计算一个 $(\beta+2)$-全范数分配。

证明。运行算法 1并配备所提供的预言机和误差参数 $\varepsilon = 1/\beta$。根据定理 2.1，结果 $\bar{z}$ 满足 $\bar{z}(\delta(c)) \ge 1-\varepsilon/2$ 对每个客户 c 以及 $\|L_{\bar{z}}\|_p \le \beta \cdot \mathrm{OPT}_p$对所有 $p \ge 1$均成立。将 $\bar{z}$ 缩放 $1+\varepsilon$，并对结果运行引理 4.1中的舍入算法，我们得到一个整数分配 $\hat{z}$，使得

$$\|L_{\hat{z}}\|_p \le (1+\varepsilon)\|L_{\bar{z}}\|_p + \|w\|_p \le (1+\varepsilon)\beta \cdot \mathrm{OPT}_p + \mathrm{OPT}_p \le (\beta+2)\cdot \mathrm{OPT}_p$$

对所有 $p \ge 1$ 均成立，证毕。

每次调用预言机都需要对数据流进行 p 遍扫描，并需要 $S(n,m)$。由于预言机的宽度为 d，乘法权重更新算法会调用预言机 $O(d^2 \log n/\varepsilon^2)$ 次。因此，总共需要进行 $O(d^2 p \log n/\varepsilon^2)$ 遍扫描。

最后，定理 2.1 所需的空间为 $O(S(n,m)\cdot d^2 \log n/\varepsilon^2 + n)$，而执行舍入算法最多只会使其增加一倍。 □

在 β-全范数预言机问题中，输入是一个二分图 $G = (C, S, E)$，每个 $c \in C$ 的权重为 $w(c)$，每个 $c \in C$ 的值为 $\lambda(c)$。目标是计算一个 $z : E \to \mathbb{R}_+$，使得

$$\sum_{c \in C} \lambda(c) z(\delta(c)) \geq \sum_{c \in C} \lambda(c), \tag{CCC}$$

and

$$\left(\sum_{s \in S} \left(\sum_{c \in N(s)} w(c) z(cs) \right)^p \right)^{1/p} \leq \beta \cdot \mathrm{OPT}_p, \tag{LPC}$$

对于所有 $p \geq 1$。如果对所有 $c \in C$，$|z(\delta(c)) - 1| \leq d$，则该算法是 d-有界的。

图 1：β-全范数预言机问题。

4.2 NestedMatching 层级

回顾一下，在全范数预言机问题（图 1）中，每个客户 c 都关联着两个量：价值 $\lambda(c)$ 和权重 $w(c)$。分配价值较大的客户更容易满足组合客户端覆盖约束（CCC），而分配权重较大的客户则会使满足 ℓ_p-范数约束（LPC）更加困难。客户 的单位权重价值 c，记作 $\rho(c)$，为 $\lambda(c)/w(c)$。令 $\varepsilon > 0$；我们将在稍后选择 ε。

定义 4.1。客户 c 的单位权重价值记为 $\rho(c)$，其定义为 $\lambda(c)/w(c)$。对于 $j \in \mathbb{Z}$，定义 $C^{(j)} = \{c \in C : \rho(c) \geq \max_{c \in C} \rho(c) \cdot e^{-\varepsilon j}\}$.。另定义 $E^{(j)} = \{cs \in E : c \in C^{(j)}\}$ 和 $G^{(j)} = (C, S, E^{(j)})$。

我们利用 $E^{(-1)} = \emptyset$ 来简化部分证明。注意，$C^{(0)}$ 是单位权重价值最大的客户集合，并且对于所有整数 j，都有 $C^{(j)} \subseteq C^{(j+1)}$。下一个简单引理指出，对于某个 $j = O(\log(\max_{c \in C} \rho(c) w(C))/\varepsilon)$，我们可以忽略 $C^{(j)}$ 之外的客户。

引理 4.3. 设 $R = \max_{c \in C} \rho(c)$。对于 $j \geq \ln(R \cdot w(C)/\varepsilon)/\varepsilon$，有 $\lambda(C^{(j)}) \geq \lambda(C) - \varepsilon$.

证明。 $\lambda(C) - \lambda(C^{(j)}) = \lambda(C \setminus C^{(j)}) = \sum_{c \in C \setminus C^{(j)}} \rho(c) w(c) \leq R e^{-\varepsilon j} w(C \setminus C^{(j)}) \leq \varepsilon$. □

分数型 $(\kappa, 2^i)$-嵌套匹配层级是由一系列 b-匹配 $x_0, x_1, \ldots, x_\ell$ 构成，其中每个 x_i 都是某个容量函数 κ 下的分数型 $(\kappa, 2^i)$-匹配，并且每个 x_i 都包含于 x_{i+1}。

定义 4.2（规模因子）。如果将分数型 b-匹配 x 限制到 $E^{(j)}$ 后，对于所有 $j \in \{0, \ldots, k\}$，它都是 $G^{(j)} = G[C^{(j)}, S]$ 的一个 α-近似 b-匹配，那么称 G 的分数型 b-匹配 x 相对于 $(C^{(j)})_{j=0}^k$ 具有规模因子 α。

定义 4.3（分数型嵌套匹配层级）。设 $\kappa : C \to \mathbb{R}_+$。设对于 $i \in \{0, \ldots, \ell\}$，$x_i : E \to \mathbb{R}_+$。序列 $\mathbf{x} = (x_0, \ldots, x_\ell)$ 是一个分数型 $(\kappa, 2^i)$-嵌套匹配层级（嵌套匹配层级），当且仅当对于所有 $i \in \{0, \ldots, \ell\}$，

1. x_i 是一个分数（κ，2^i）-匹配，且

2. 对于所有 $e \in E$， $x_i(e) \geq x_{i-1}(e)$。

我们称 $\mathbf{x}$ 关于 $(C^{(j)})_{j=0}^{k}$ 具有规模因子 α，如果每个 x_i 关于 $(C^{(j)})_{j=0}^{k}$都具有规模因子 α。

计算一个（w， 2^i）-嵌套匹配层级，其关于（$C^{(j)}$）$_{j=0}^{k}$ 的规模因子为 8，即使在半流式设置下，也相对简单。该算法归结为计算 $\operatorname{polylog} n$ 个极大 b-匹配。具体细节留待第 4.5 节。

4.3 匹配层次结构与 ℓ_p-范数

给定一个分数型 $(w, 2^i)$-嵌套匹配层级 $\mathbf{x} = (x_0, \ldots, x_\ell)$，我们可以根据其他匹配$x_i$，对匹配 x_ℓ 的负载向量的 ℓ_p-范数得到一个相当好的上界。

引理 4.4. 如果 $\mathbf{x} = (x_0, \ldots, x_\ell)$ 是一个分数型 $(w, 2^i)$-嵌套匹配层级，那么

$$\sum_{s \in S} x_\ell(\delta(s))^p \leq p \cdot x_0(E) + \sum_{i=1}^{\ell} p 2^{i(p-1)} \cdot (x_i - x_{i-1})(E).$$

证明。固定 $s \in S$。对于 $i \in \{-1, \ldots, \ell\}$，令 $d_i = x_i(\delta(s))$。令 $g\colon \mathbb{R} \to \mathbb{R}$ 为函数$g(z) = z^p$，并回忆 $g'(z) = pz^{p-1}$。由于 g 可微，

$$g(x_\ell(\delta(s))) = \sum_{i=0}^{\ell} g(d_i) - g(d_{i-1}) = \sum_{i=0}^{\ell} \int_{d_{i-1}}^{d_i} g'(z) dz.$$

利用 g 是凸的以及 $d_i \leq 2^i$，

$$\int_{d_{i-1}}^{d_i} g'(z) dz \leq g'(d_i)(d_i - d_{i-1}) \leq g'(2^i)(d_i - d_{i-1}). \tag{5}$$

因此

$$x_\ell(\delta(s))^p \leq \sum_{i=0}^{\ell} p 2^{i(p-1)} (x_i - x_{i-1})(\delta(s)).$$

对所有 $s \in S$ 求和即可得到结论。 □

我们还为结构更复杂的匹配层次结构证明一个互补的下界。首先需要以下定义。

定义 4.4。若对于所有 $s \in S$ 和 $i \in \{0, \ldots, \ell - 1\}$，当 $x_{i+1}(\delta(s)) \neq x_i(\delta(s))$ 时，$x_i(\delta(s)) = 2^i$，则分数型 $(\kappa, 2^i)$-嵌套匹配层级（NMH）$(x_0, \ldots, x_\ell)$ 是极大嵌套的。

极大嵌套层次结构具有极大性，其含义是：对于所有 $i \in \{0, \ldots, \ell - 1\}$ 和所有 $e \in E$，都不能增加 $x_i(e)$，否则会违反嵌套匹配层级的包含性质。极大嵌套整数匹配层次结构具有以下性质，我们将在下一个证明中用到：对于所有 $i \in \{0, \ldots, \ell\}$ 和 $s \in S$，数量 $(x_i - x_{i-1})(\delta(s))$ 要么为零，要么介于 1 和 2^{i-1} 之间。

引理 4.5. 若 x $=(x_0,\ldots,x_\ell)$ 是一个极大嵌套的整数 $(w,2^i)$-嵌套匹配层级，则

$$\sum_{s\in S} x_\ell(\delta(s))^p \geq x_0(E) + \sum_{i=1}^{\ell} p2^{(i-1)(p-1)} \cdot (x_i - x_{i-1})(E).$$

证明。 证明与 引理 4.4的证明类似， $s \in S$ 并令 $d_i = x_i(\delta(s))$ 对于 $i \in \{0,\ldots,\ell\}$。由于 g是可微的，

$$g(x_\ell(\delta(s))) = g(d_0) + \sum_{i=1}^{\ell} g(d_i) - g(d_{i-1}) = g(d_0) + \sum_{i=1}^{\ell} \int_{d_{i-1}}^{d_i} g'(z)dz.$$

利用 g 是凸的这一点，

$$\int_{d_{i-1}}^{d_i} g'(z)dz \geq g'(d_{i-1}) \cdot (d_i - d_{i-1}). \tag{6}$$

现在，由于 x 是极大嵌套的，且 g' 是递增的， $g'(d_{i-1}) \cdot (d_i - d_{i-1}) \geq g'(2^{i-1}) \cdot (d_i - d_{i-1})$；事实上， $d_i = d_{i-1}$ 或 $d_{i-1} = 2^{i-1}$。此外，由于 x 的整数性，我们有$d_0 \in \{0,1\}$，且无论哪种情况， $g(d_0) = d_0$。因此我们得到

$$g(x_\ell(\delta(s))) \geq g(d_0) + \sum_{i=1}^{\ell} g'(d_{i-1}) \cdot (d_i - d_{i-1}) \geq d_0 + \sum_{i=1}^{\ell} g'(2^{i-1}) \cdot (d_i - d_{i-1}).$$

对 $s \in S$ 求和即可得到所需结果。 □

我们以将一个分配简单分解为极大嵌套匹配层级来结束本小节。当我们想要将上述下界应用于一个使 ℓ_p-范数最小化的分配时，这将很有用。

定义 4.5。设 $A \subseteq E$ 是一个分配，并设 $\ell = \lceil \log_2(w(C)) \rceil$。若 x 是极大嵌套的，则 $(w,2^i)$-嵌套匹配层级 x $=(x_0,\ldots,x_\ell)$ 是 A 的 $(w,2^i)$-嵌套匹配层级分解；否则， $x_\ell(cs) = \begin{cases} w(\\ \end{cases}$

c $cs \in A$)。

0 每个分配 A 都有一个分数型 $(w,2^i)$-NMH 分解，构造如下。按照定义初始化 x_ℓ，并对所有 $i < \ell$ 和 $e \in E$ 设置 $x_i(e) = 0$。然后，当存在某个 $e \in E$ 和 $i < \ell$，使得 $x_i(e) < x_{i+1}(e)$ 且 $x_i(e) < 2^i$ 时，设置 $x_i(e) = \min\{x_{i+1}(e)\ 2^i\}$。过程终止后，所得 NMH 根据定义是极大嵌套的。

4.4 基于嵌套匹配层级的 All-NormOracle

现在，我们描述如何在给定一个计算嵌套匹配层级的算法后，求解全范数预言机问题（图 1）。我们首先计算一个关于 $(w,2^i)$-NMH y $=(y_0,\ldots,y_\ell)$ 规模因子为 α 的分数型嵌套匹配层级。$C^{(j)})_{j=0}^{k}$计算 NMH y 是唯一需要访问 G的步骤；算法的其余部分完全基于 y。

得到 y 后，算法会为所有 $e \in \text{supp}(y_\ell)$ 初始化 $x(e) = 0$。算法的主外层循环将迭代器 j 从 0 递增至 k。在第 j 次迭代中，只有那些客户

位于 $C^{(j)}$ 中的客户才会被纳入考虑。通过这种方式，外层循环隐式地维护一个较大的单位权重价值阈值；该阈值从能够产生非空客户集的最大阈值开始，并在每一步中按 $e^{\varepsilon} \approx 1+\varepsilon$ 的倍数递减。

在每次外层循环迭代中，内层循环都会递增一个迭代器 i，使其遍历匹配层级 $\mathbf{y} = (y_0, \ldots, y_\ell)$ 的各个层级。在内层循环的第 i 次迭代中，算法尝试将 y_i 在 $E^{(j)}$ 中各边上的值复制到 x 中。不过，算法不允许 x 增长得太快、过于庞大；它会在整个外层迭代 j 期间强制满足 $x(E^{(j)}) \le \frac{1}{\alpha} w(C^{(j)})$。更准确地说，对于任意满足 $x(e) < y_i(e)$ 的边 $e \in E^{(j)}$，算法会持续增加 $x(e)$，直到满足 $x(e) = y_i(e)$ 或 $\alpha x(E^{(j)}) = w(C^{(j)})$。

在外层循环结束时，计算最终结果只需进行缩放。算法设置 $z(cs) = (1+\varepsilon)^2 \alpha x(cs)/w(c)$ 对于 $cs \in$ 支撑集(x) 并返回 z。参见 算法 2。

算法 2： 一个 $O(1)$-全范数预言机。

输入：图 $G = (C, S, E)$，客户权重 $w\colon\ C \to \mathbb{Z}_+$，客户价值 $\lambda\colon\ C \to \mathbb{R}_+$，以及精度参数 $\varepsilon > 0$。

1. 设置 $R \leftarrow \max_{c \in C} \rho(c)$。
2. 设置 $\ell \leftarrow \lceil \log_2 n \rceil$；设置 $k \leftarrow \lceil \log(Rw(C)/\varepsilon)/\varepsilon \rceil$。
3. //这些量 R 和 ε 定义了集合 $C^{(j)}$。
4. 计算一个关于 $(C^{(j)})_{j=0}^{k}$ 的分数 $(w, 2^i)$-嵌套匹配层级 $\mathbf{y} = (y_0, \ldots, y_\ell)$，其规模因子为 α。
5. //图的边 E 仅在上一步中用于计算$\mathbf{y}$。
6. 设置 $x(e) \leftarrow 0$ 对于 $e \in$ 支撑集(y_ℓ)。
7. 对于 $j \leftarrow 0$ 至 k，
8. 　对于 $i \leftarrow 0$ 至 ℓ，
9. 　　当 $x(E^{(j)}) < \frac{1}{\alpha} w(C^{(j)})$，并且存在某个 $e \in E^{(j)}$ 使得 $x(e) < y_i(e)$，将 $x(e)$ 增加最小的量，使得以下条件之一成立：$x(E^{(j)}) = \frac{1}{\alpha} w(C^{(j)})$ 或 $x(e) = y_i(e)$。
10. 对于 $cs \in \mathrm{supp}(x)$，设置 $z(cs) \leftarrow (1+\varepsilon)^2 \, \alpha \cdot x(cs)/w(c)$。
11. 返回 z。

预言机的正确性。现在开始证明算法 2解决了全范数预言机问题（图 1）。需要证明三点：

1. 输出 z 满足组合客户端覆盖约束（CCC）。2. 输出 z 满足 β-近似 ℓ_p-范数约束，其中常数 β（LPC）。

3. 输出 z 的宽度为 $O(1)$。

W我们先给出关于 算法 2 的定义，然后收集一些有用的观察结论，其证明显而易见。

定义 4.6。 外层循环 是从第 7 行开始的循环，而内层循环 是从第 第 8 行开始的循环。我们用外层迭代 j 表示外层循环的第 j 次迭代，用内层迭代 (j, i) 表示在外层迭代 i 期间内层循环的第 j 次迭代。

观察 4.6。令 $j \in \{0,\dots,k\}$。在外层循环的第 j 次迭代结束时，x 由 $E^{(j)}$ 支撑。

观察 4.7。设 $j \in \{0,\dots,k\}$。在外层循环的整个第 j 次迭代中，$x(E^{(j)}) \leq \frac{1}{\alpha} w(C^{(j)})$。

观察 4.8。在整个算法过程中，对于所有 $e \in E$，$x(e)$ 都单调增加。

观察 4.9。算法结束时，对于所有 $e \in E$，$x(e) \leq y_\ell(e)$。

观察 4.10。在内层迭代 (j,i) 结束时，对于所有 $e \in E^{(j)}$，要么 $x(E^{(j)}) = \frac{1}{\alpha} w(C^{(j)})$，要么 $x(e) \geq y_i(e)$。

观察 4.11。在算法执行的任何时刻，$x(\delta(c)) \leq w(c)$。

现在我们证明，观察 4.7 中的不等式实际上在外层迭代结束时取等号 j。

引理 4.12. 设 $j \in \{0,\dots,k\}$。在外层循环第 j 次迭代结束时，$x(E^{(j)}) = \frac{1}{\alpha} w(C^{(j)})$。

证明。设 $j \in \{0,\dots,k\}$。根据 观察 4.10，在外层迭代 j结束时——也就是在内层迭代 (j,ℓ) 结束时——要么 $x(E^{(j)}) = \frac{1}{\alpha} w(C^{(j)})$，要么 $x(e) \geq y_\ell(e)$ 对所有 $e \in E^{(j)}$成立。因此，只需证明 $y_\ell(E^{(j)}) \geq \frac{1}{\alpha} w(C^{(j)})$。

回忆一下，y_ℓ是一个规模因子为 α 的 $(w, 2^\ell)$-匹配。最大 $(w, 2^\ell)$- 匹配的大小显然为 $w(C)$。事实上，由于 $2^\ell = 2^{\lceil \log_2 w(C) \rceil} \geq w(C)$，$(w, 2^\ell)$-匹配的服务器容量非常大，因此可以任意匹配每个客户，而不会违反任何服务器容量。因此，$\alpha \cdot y_\ell(E^{(j)}) \geq w(C) \geq w(C^{(j)})$。□

下一个引理证明，预言机的输出满足组合客户端覆盖约束。

引理 4.13。输出 z 属于 算法 2 满足 $\sum_{c \in C} \lambda(c) z(\delta(c)) \geq \sum_{c \in C} \lambda(c$).

证明。只需证明在算法结束时

$$(1+\varepsilon)\alpha \sum_{c \in C^{(k)}} \rho(c) x(\delta(c)) \geq \sum_{c \in C^{(k)}} \rho(c) w(c). \tag{7}$$

事实上，若 (7) 成立，则由于 x 支撑在 $E^{(k)}$，

$$\begin{aligned}
\sum_{c \in C} \lambda(c) z(\delta(c)) &= (1+\varepsilon)^2 \alpha \sum_{c \in C} \rho(c) x(\delta(c)) \\
&= (1+\varepsilon)^2 \alpha \sum_{c \in C^{(k)}} \rho(c) x(\delta(c)) \\
&\geq (1+\varepsilon) \sum_{c \in C^{(k)}} \rho(c) w(c) = (1+\varepsilon) \sum_{c \in C^{(k)}} \lambda(c) \geq \sum_{c \in C} \lambda(c),
\end{aligned}$$

其中最后一个不等式由 引理 4.3 得出。

现在来证明不等式（7）。令 $r_j = e^{-\varepsilon j} \cdot \max_{c\in C} \rho(c)$ 为定义 $C^{(j)}$ 的量。对于任意函数 $f\colon C \to \mathbb{R}_+$，我们可以用黎曼型求和给 $\sum_{c\in C} f(c)\rho(c)$ 建立上下界：

$$\sum_{j=0}^{k} f(C^{(j)})(r_j - r_{j+1}) \le \sum_{c\in C^{(k)}} f(c)\rho(c) \le \sum_{j=0}^{k} f(C^{(j)})(r_{j-1} - r_j). \tag{8}$$

根据引理 4.12，$x(E^{(j)}) = \frac{1}{\alpha} x(\delta(C^{(j)}))$ 在外层循环第 j 次迭代结束时成立。由于x 单调增加，因此算法结束时，$x(E^{(j)}) \ge \frac{1}{\alpha} x(\delta(C^{(j)}))$ 对所有 $j \in \{0,\dots,k\}$均成立。综上，

$$\begin{aligned}
\sum_{c\in C^{(k)}} w(c)\rho(c) &\le \sum_{j=0}^{k} w(C^{(j)})(r_{j-1} - r_j) \\
&\le \alpha \sum_{j=0}^{k} x(E^{(j)})(r_{j-1} - r_j) \\
&\le (1+\varepsilon)\alpha \sum_{j=0}^{k} x(E^{(j)})(r_j - r_{j+1}) \\
&\le (1+\varepsilon)\alpha \sum_{c\in C^{(k)}} x(\delta(c))\rho(c). \qquad \square
\end{aligned}$$

在证明组合客户端覆盖约束得到满足后，我们现在继续证明 ℓ_p-范数约束得到满足。可以将下述引理中的 $(x_i^*)_i$ 视为某个 $(w, 2^i)$-NMH 分解，该分解对应于一个 ℓ_p-范数最优分配（回顾定义 4.5），事实上，我们稍后正是在本小节中以此方式应用它。

引理 4.14。设 $(x_0^*,\dots,x_\ell^*)$ 是一个分配的分数型 $(w, 2^i)$-NMH 分解，并令 $x^* = x_\ell^*$。在算法 2结束时，

$$\alpha \cdot (x - y_i)^+(E) \le (x^* - x_i^*)(E),$$

对于所有 $i \in \{0,\dots,\ell\}$。

证明。令 $x^{(j)}$ 表示 x 在外层循环第 j 次迭代结束时的值。等价地，$x^{(j)}$ 是 x 在外层循环第 $j+1$ 次迭代开始时的值，因此我们也将 $x^{(-1)}$ 定义为零向量。

令 $P(j)$ 表示以下命题：

$$\alpha \cdot (x^{(j)} - y_i)^+(E^{(j)}) \le (x^* - x_i^*)(E^{(j)}) \text{ for all } i \in \{0,\dots,\ell\}. \tag{9}$$

我们通过归纳法证明，对所有 $j \in \{-1,\dots,k\}$，$P(j)$ 都成立。但首先，我们来看看为什么 $P(k)$ 蕴含引理的结论。算法结束时 x 的值也称为 $x^{(k)}$，因此，由于 $P(k)$ 成立，我们有

$$\alpha \cdot (x^{(k)} - y_i)^+(E^{(k)}) \le (x^* - x_i^*)(E^{(k)})$$

对于所有 i。左侧的量与 $\alpha\cdot(x^{(k)} - y_i)^+(E)$ 相同，因为 $x^{(k)}$ 的支撑集包含于 $E^{(k)}$。右侧的量显然至多为 $(x^* - x_i^*)(E)$，因为 $E^{(k)} \subseteq E$。

现在我们回到证明对于所有 $j \in \{-1, \dots, k\}$， $P(j)$ 成立。固定 j。如果 $j = -1$，则该结论显然成立。否则， $j \geq 0$，我们可以归纳假设 $P(j-1)$ 成立。在外层迭代 j 期间，有两种内层迭代 i: 一种以 $x(E^{(j)}) < \frac{1}{\alpha} w(C^{(j)})$ 结束（因此 $x \geq y_i$），另一种因 $x(E^{(j)}) = \frac{1}{\alpha} w(E^{(j)})$ 而结束。

情形 1：在内层迭代 (j, i) 结束时， $x(E^{(j)}) < \frac{1}{\alpha} w(C^{(j)})$。此情形不需要归纳假设。在内层迭代 (j, i) 结束时，由于 $x(E^{(j)}) < \frac{1}{\alpha} w(C^{(j)})$，对于所有 $e \in E^{(j)}$，必有 $x(e) \geq y_i(e)$（否则迭代就不会完成）。根据单调性，对于所有 $e \in E^{(j)}$，有 $x^{(j)}(e) \geq y_i(e)$。因此

$$\alpha \cdot (x^{(j)} - y_i)^+(E^{(j)}) = \alpha \cdot (x^{(j)} - y_i)(E^{(j)}) = \alpha x^{(j)}(E^{(j)}) - \alpha y_i(E^{(j)}).$$

根据引理 4.12，我们有 $\alpha x^{(j)}(E^{(j)}) = w(C^{(j)})$；根据 x^* 的定义（见定义 4.5），有 $w(C^{(j)}) = x^*(E^{(j)})$。最后，注意到 $\alpha y_i(E^{(j)}) \geq x_i^*(E^{(j)})$，因为 y_i 是一个规模因子为 α 的 $(w, 2^i)$-匹配，而 x_i^* 是一个 $(w, 2^i)$-匹配。

情形 2：在内层迭代 (j, i) 结束时， $x(E^{(j)}) = \frac{1}{\alpha} w(C^{(j)})$。我们论证

$$(x^{(j)} - y_i)^+(e) = (x^{(j-1)} - y_i)^+(e) \text{ for all } e \in E^{(j)}. \tag{10}$$

首先来看为什么这足以证明 $P(j)$，假设 $P(j-1)$。我们有

$$\begin{aligned}
(x^{(j)} - y_i)^+(E^{(j)}) &= (x^{(j-1)} - y_i)^+(E^{(j)}) && \text{(by (10))} \\
&= (x^{(j-1)} - y_i)^+(E^{(j-1)}) && \text{(because } x^{(j-1)} \text{ is supported on } E^{(j-1)}) \\
&\leq (x^* - x_i^*)(E^{(j-1)}) && \text{(by } P(j-1)) \\
&\leq (x^* - x_i^*)(E^{(j)}).
\end{aligned}$$

现在来证明 (10)。设 $e \in E^{(j)}$。如果 $x(e)$ 在外层迭代j中完全没有增加，那么结论成立。因此，假设 $x(e)$ 在外层迭代 j 中增加，并令 i^* 为满足 $x(e)$ 在内层迭代 (j, i^*) 中增加的最大整数。由于在内层迭代 (j, i) 结束时，我们有 $x(E^{(j)}) = \frac{1}{\alpha} w(C^{(j)})$（根据假设），因此对于满足 $i' > i$ 的任何内层迭代 (j, i')， x 都不会增加。由此可得 $i^* \leq i$。

由于内层迭代 (j, i^*) 是外层迭代 j 中最后一个使 $x(e)$ 增加的内层迭代，且它使 $x(e)$ 至多增加到 $y_{i^*}(e)$，因此我们有 $x^{(j)}(e) \leq y_{i^*}(e)$。由于 y 是一个嵌套匹配层级，且 $i^* \leq i$，因此我们有 $y_{i^*}(e) \leq y_i(e)$。于是 $(x^{(j)} - y_i)^+(e) = 0$，并且根据单调性，$(x^{(j-1)} - y_i)^+(e) = 0$ 也成立。 □

引理 4.15。 令 z 为 算法 2的输出。对于所有 $p \geq 1$,

$$\left(\sum_{s \in S} \left(\sum_{c \in N(s)} w(c) z(cs) \right)^p \right)^{1/p} \leq \frac{2^{1-1/p}}{\alpha^{1/p}} \cdot \mathrm{OPT}_p.$$

证明。固定 $p \geq 1$。在算法结束时，我们有

$$\sum_{s \in S} \left(\sum_{c \in N(s)} w(c) z(\delta(s)) \right)^p = \sum_{s \in S} x(\delta(s))^p.$$

定义 $x_i = \min\{x, y_i\}$ 对于 $i \in \{0, \dots, \ell\}$。为使记号更简洁，我们再定义 $a_i = p2^{i(p-1)}$ 对于 $i \in \{0, \dots, \ell\}$。根据 引理 4.4,

$$\begin{aligned}\sum_{s\in S} x(\delta(s))^p &\le a_0 x_0(E) + \sum_{i=1}^{\ell} a_i (x_i - x_{i-1})(E) \\ &= a_0 x_\ell(E) + \sum_{i=1}^{\ell} (a_i - a_{i-1})(x_\ell - x_{i-1})(E).\end{aligned}$$

Now 令 A^* 为使 ℓ_p-范数最小化的分配，并令 $\mathbf{x}^* = (x_0^*, \dots, x_\ell^*)$ 为一个最大 -ima层嵌套（w, 2^i）-NMH 分解，用于 A^*。根据 引理 4.5,

$$\begin{aligned}\sum_{s\in S} x^*(\delta(s))^p &\ge x_0^*(E) + \sum_{i=1}^{\ell} a_{i-1}(x_i^* - x_{i-1}^*)(E) \\ &= x_\ell^*(E) + (a_0 - 1)(x_\ell^* - x_0)(E) + \sum_{i=2}^{\ell} (a_{i-1} - a_{i-2})(x_\ell^* - x_{i-1}^*)(E).\end{aligned}$$

上界和下界几乎相等；我们可以使用 引理 4.14 将其中一个转换为另一个。回顾一下，引理 4.14指出

$$\alpha \cdot (x_\ell - x_{i-1})(E) \le (x_\ell^* - x_{i-1}^*)(E).$$

因此

$$\begin{aligned}\sum_{s\in S} x(\delta(s))^p &\le a_0 x_\ell(E) + \sum_{i=1}^{\ell} (a_i - a_{i-1}) \cdot (x_\ell - x_{i-1})(E) \\ &\le \frac{1}{\alpha}\left(a_0 x_\ell^*(E) + \sum_{i=1}^{\ell} (a_i - a_{i-1})(x_\ell^* - x_{i-1}^*)(E)\right) \\ &\le \frac{2^p}{\alpha}\left(x_\ell^*(E) + (a_0 - 1)(x_\ell^* - x_0)(E) + \sum_{i=2}^{\ell} (a_{i-1} - a_{i-2})(x_\ell^* - x_{i-1}^*)(E)\right) \\ &\le \frac{2^p}{\alpha} \sum_{s\in S} x^*(\delta(s))^p = \frac{2^p}{\alpha} \cdot \mathrm{OPT}_p^p.\end{aligned}$$

□

引理 4.16。 对于每个 $\varepsilon > 0$, 算法 2 都会为 $(1+\varepsilon)^2 2\alpha^{1-1/p}$-全范数预言机问题生成一个解。

证明。 令 z 为算法 2 的输出，并令 x 为算法结束时预言机计算出的量。由于 $z(cs) = (1+\varepsilon)^2 \alpha x(cs)/w(c)$ 对于 $cs \in \mathrm{supp}(x)$，我们有

$$\sum_{c\in C} \lambda(c) z(\delta(c)) = (1+\varepsilon)^2 \alpha \sum_{c\in C} \rho(c) x(\delta(c)).$$

引理 4.13 恰好说明组合客户端覆盖约束得以满足。根据引理 4.15,

$$\begin{aligned}\left(\sum_{s\in S}\left(\sum_{c\in N(s)} w(c) z(cs)\right)^p\right)^{1/p} &= (1+\varepsilon)^2 \alpha \cdot \left(\sum_{s\in S} x(\delta(s))^p\right)^{1/p} \\ &\le (1+\varepsilon)^2 (2\alpha)^{1-1/p} \cdot \mathrm{OPT}_p.\end{aligned}$$

最后，由于对于所有 $c \in C$， $x(\delta(c)) \leq w(c)$，因此对于所有 $c \in C$， $z(\delta(c)) \leq (1+\varepsilon)^2\alpha$。 □

引理 4.17。如果存在一种半流式算法，能够使用 p 遍计算规模因子为 α 的 $(w, 2^i)$-嵌套匹配层级，那么就存在一种半流式算法，能够计算 $(1+\varepsilon)2\alpha$-全范数分数分配。

证明。 运行 算法 2，使用该半流式算法实现 第 4 行。嵌套匹配层级 $\mathbf{y}$ 的输出支撑集大小必须为 $O(n$多对数 $n)$。 算法 2 所使用的额外空间用于计算 x 和 z，它们与 y_ℓ具有相同的支撑集。因此，整个算法使用 $O(n$多对数 $n)$ 空间。

根据 引理 4.16，输出是一个 $(1+\varepsilon)(2\alpha+1)$-全范数分配。 □

4.5 半流式模型中的嵌套匹配层级

我们的任务已归结为在半流式环境中计算关于 $(C^{(j)})_{j=0}^{k}$ 的嵌套匹配层级。

计算 b-匹配，其规模因子为 4。 Crouch 和 Stubbs [20] 描述了一种单遍算法，用于使用任意 α-近似最大基数匹配算法计算规模因子为 2α 的简单匹配（即整数型且单位容量）。他们的算 b-匹配情形，并且因此任意 α-近似最大基数 b-匹配算法都可用于计算规模因子为 b 的 2α-匹配。

主要子过程是一个贪心算法 MergeInto(x, y)，它以两个b-匹配 x 和 y 作为输入，并返回一个 b-匹配 $\tilde{x}$，该匹配（1）包含 x，且（2）其大小至少为 y大小的一半。该算法通过从 y 中贪心地（可能以分数形式）向 x 添加边来运行，同时满足容量约束 b。

引理 4.18. 设 x 和 y 是 b-匹配。输出 $\tilde{x} \leftarrow$ MergeInto(x, y) 是一个 b-匹配，满足对所有 $e \in E$， $\tilde{x}(e) \geq x(e)$，且 $2\tilde{x}(E) \geq y(E)$。

我们省略前一个引理的证明；该证明与以下结论的证明相似：最大匹配的大小 f a m至少是最大匹配大小的一半。

我们使用贪心 MergeInto 过程，结合 α-近似最大基数匹配算法，来计算一个 b-匹配，其规模因子为 2α。该算法如下运行。首先，它为每个子图 α 计算一个近似 b-匹配 $x^{(j)}$。然后，它设置 $\tilde{x}^{(1)} \leftarrow x^{(1)}$，并对每个 $j \geq 1$，设置 $\tilde{x}^{(j)} \leftarrow$ MergeInto$(\tilde{x}^{(j-1)}, x^{(j)})$.最后，它返回 $\tilde{x}^{(k)}$。该算法总结于 算法 3。

引理 4.19. 由 $\tilde{x}^{(k)}$ 返回的 算法 3 是一个 b-匹配，其规模因子为 2α。

证明。 设 $j \in \{0, \ldots, k\}$ 并设 y是一个 b-匹配。根据 引理 4.18 以及简单的归纳法，$\tilde{x}^{(k)}$ 包含 $\tilde{x}^{(j)}$。现在我们有

$$\begin{aligned} 2\alpha\tilde{x}^{(j)}(E^{(j)}) &\geq \alpha x^{(j)}(E^{(j)}) && \text{(by Lemma 4.18)} \\ &= \alpha x^{(j)}(E) && (\mathrm{supp}(x^{(j)}) \subseteq E^{(j)}) \\ &\geq y(E^{(j)})。 && (x^{(j)} \text{ is } \alpha\text{-approximate in } G^{(j)}) \end{aligned}$$ □

引理 4.20. 如果存在一个 p 遍、使用 $S(n,m)$ 空间的流式算法，可计算一个 α-近似的 b-匹配，那么就存在一个 p 遍、使用 $O(S(n,m) \cdot k)$ 空间的流式算法，可计算一个相对于 $(C^{(j)})_{j=0}^{k}$ 具有 2α 规模因子的 b- 匹配。

算法 3：计算规模因子为 4 的 (κ, τ)-匹配。

输入 ：一个用于遍历 G 的边的迭代器、客户端容量 κ 和服务器容量 τ.

输出：一个规模因子为 4 的 (κ, τ)-匹配。

1. MergeInto(x, y):
2. 对每个 $e \in \text{supp}(x) \cup \text{supp}(y)$，设置 $\tilde{x}(e) \leftarrow x(e)$。
3. 对于 $cs \in$ 支撑集(x),
4. 将 $\tilde{x}(cs)$ 增加最小的幅度，使得以下条件之一成立： $\tilde{x}(\delta(c)) = \kappa(c)$, $\tilde{x}(\delta(s)) = \tau(s)$, 或 $\tilde{x}(cs) = x(cs) + y(cs)$】【。
5. 返回 $\tilde{x}$。
6. 对于每个 $j \in \{0, \ldots, k\}$，使用一个计算 $G^{(j)}$ 的极大 (κ, τ)-匹配 $x^{(j)}$。
   遍历边数据流。
7. 设置 $\tilde{x}^{(0)} \leftarrow x^{(0)}$。
8. 对于 $j \leftarrow 1$ 到 k，设置 $\tilde{x}^{(j)} \leftarrow \text{MergeInto}(\tilde{x}^{(j-1)}, x^{(j)})$。
9. 返回 $\tilde{x}^{(k)}$。

证明。 算法 3算法 3的第一步，第 6 行，可以在半流式设置中用 p 遍实现，方法如下。算法首先为每个 A_j 对应的 $G^{(j)}$ 实例化一个流式算法 $j \in \{0, \ldots, k\}$。每当一条边 e（在数据流的 p遍中）到达时，就将 e 发送给每个满足 A_j 的 $e \in G^{(j)}$。经过 p 遍后，每个算法 A_j 返回 $x^{(j)}$。

算法的下一步，即合并步骤，直接根据 $x^{(j)}$ 计算，无需再次遍历数据流。使用的总空间为 $O(S(n, m) \cdot k)$。 □

与极大匹配可以通过单遍计算且为 2-近似一样，极大分数 b-匹配也可以通过单遍计算且为 2-近似。

引理 4.21。存在一种单遍 $O(n\log\ n)$-空间算法，可用于计算 2-近似分数 b-匹配。

证明。该算法维护一个分数 b-匹配 x，初始为空。在每条边 e 到达时，它将 $x(e)$ 设置为在满足容量约束的前提下尽可能大的值。如果 $x(e)$ 发生任何变化，此步骤总会使其中一个端点达到饱和。因此，在算法结束时， x 的支撑集是 $O(n)$。

事实是 x 是一个 2-近似，这属于众所周知的结论，也可以由 引理 4.18推出：如果x^* 是一个最大的分数 b-匹配，则输出 $\tilde{x}$ MergeSize(x, x^*) 包含 x，且大小至少为 x^*的一半。由于 x 是极大的， $\tilde{x}$ 必须等于 x。 □

作为 引理 4.20 和 4.21的一个简单推论，我们可以在一次遍历中计算出一个规模因子为 4 的 b-匹配。正如前文所述，这与 Crouch 和 Stubbs 的算法类似， [20], 但针对的是分数 b-匹配，而不是简单匹配。

推论 4.21.1. 存在一种单遍 $O(n\log\ n{\cdot}k)$-空间流式算法，可计算规模因子为 4 的 a b-匹配。

我们现在说明如何利用计算规模因子为 α 的 b-匹配的算法，计算规模因子为 2α 的分数型 $(\kappa, 2^i)$-嵌套匹配层级。基本子程序 HMergeInto,

以一个 (κ,τ)-匹配 x 和一个 $(\kappa,2\tau)$-匹配 y 作为输入，并返回一个 $(\kappa,2\tau)$-匹配$\tilde{x}$，使得 (1) $\tilde{x}$ 包含 x，且 (2) 对所有 $j\in\{0,\dots,k\}$，$2\tilde{x}(E^{(j)})\geq y(E^{(j)})$。该算法的工作方式与 MergeInto 非常相似，只不过它尝试按照密度递减的顺序合并 y 的边；参见算法 4。下一个引理总结了 HMergeInto 的关键性质。

引理 4.22。设 x 是一个分数 (κ,τ)-匹配，y 是一个分数 $(\kappa,2\tau)$-匹配。HMergeInto(x, y, 2τ) 的输出 $\tilde{x}$ 满足

1. $\tilde{x}$ 是一个 $(\kappa,2\tau)$-匹配；2. $\tilde{x}$ 包含 x；
3. 对所有 $j\in\{0,\dots,k\}$，$2\tilde{x}(E^{(j)})\geq y(E^{(j)})$。

特别是，如果 y 的规模因子为 α，那么 $\tilde{x}$ 的规模因子为 2α。

证明引理 4.22是常规的，不过有些繁琐。我们将其推迟到论文的完整版中给出。

算法 4：计算规模因子为 8 的 $(\kappa,2^i)$-嵌套匹配层级。

输入 ：对 G 的边、客户端容量 κ和服务器容量进行迭代的迭代器 τ.
输出：一个规模因子为 4 的 $(\kappa,2^i)$-嵌套匹配层级。

```
HMergeInto(x, y, τ):
    对每个 e ∈ supp(x) ∪supp(y)，设置 x̃(e) ← x(e)。
    对于 j ← 0 到 k,
        对于 cs ∈ supp(y) ∩ (E^(j) \ E^(j−1)),
            将 x̃(cs) 增加最小量，使得 x̃(δ(c)) = κ(c),
                x̃(δ(s)) = τ(s)，或 x̃(cs) = x(cs) + y(cs)。
    返回 x̃。
使用一遍数据流计算一个大小因子为 4 的 (κ, 2^i)-匹配 x_i，针对 i ∈ {0,...,ℓ} 边的数据流。
设置 x̃_0 ← x_0。
对于 i ← 1 到 ℓ，设置 x̃_i ← HMergeInto(x̃_{i−1}, x_i, 2^i)。
返回 x̃ = (x̃_0, ..., x̃_ℓ)。
```

下一个引理证明 算法 4 是正确的，并且可以在半流式设置中实现。

引理 4.23。设 $\kappa: C\to\mathbb{R}_+$。如果存在一个 p 遍 $S(n,m)$ 空间的流式算法，用于计算规模因子为 α 的分数 b 匹配，那么就存在一个 p 遍 $O(S(n,m)\cdot\ell)$ 空间的流式算法，用于计算相对于 $(C^{(j)})_{j=0}^{k}$ 的规模因子为 2α 的分数 $(\kappa,2^i)$-嵌套匹配层级。

证明。使用给定的算法，需要 p 遍 $O(S(n,m)\cdot\ell)$ 空间来运行 第 7 行 的算法 4。其他步骤不需要遍历数据流，且额外使用的空间也为 $O(S(n,m)\cdot\ell)$。

尚需证明，数据流返回的价值 $\tilde{\mathbf{x}}$ 是一个规模因子为 2α的分数型（κ，2^i）嵌套匹配层级，这可由引理 4.22迅速推出。首先，我们有 $\tilde{x}_0$ 等于

于 x_0，因此 $\tilde{x}_0$ 是规模因子为 α 的分数型（1，1）匹配。对于 $i \in \{1, \ldots, \ell\}$，我们有 $\tilde{x}_i = \text{HMergeInto}(\tilde{x}_{i-1}, x_i, 2^i)$，因此引理 4.22 指出，$\tilde{x}_i$ 包含 $\tilde{x}_{i-1}$，且 $\tilde{x}_i$ 的规模因子为 2α。因此（$\tilde{x}_0$，……，$\tilde{x}_\ell$）是一个分数型（1，2^i）嵌套匹配层级。 □

本小节的结论如下。

推论 4.23.1. 存在一个单遍 $O(n \cdot \ell \cdot k)$ 空间流式算法，可计算一个大小因子为 8、相对于 $(C^{(j)})_{j=0}^{k}$ 的分数 $(\kappa, 2^i)$-嵌套匹配层级 $(x_0, \ldots, x_\ell)$。

最后给出我们的主要定理。

定理 4.24. 存在一个 $O(\log n)$ 遍半流式算法，可计算一个 19-全范数指派。

证明。根据推论 4.23.1，存在一个单遍半流式算法，可计算一个相对于 $(w, 2^i)$ 的分数嵌套匹配层级，大小因子为 $8C^{(j)})_{j=0}^{k}$。取 $\varepsilon = 1/16$，由引理 4.17 可知，存在一个半流式算法，可计算一个 17-全范数分数分配。进而由引理 4.2 可知，存在一个 $O(\log n)$-遍半流式算法，可计算一个 19-全范数指派。 □

参考文献

[1] Kook Jin Ahn 和 Sudipto Guha。在半流式模型中进行线性规划及其在最大匹配问题中的应用。载于第38届自动机、语言与编程国际研讨会，526–538页，2011年。1，2[2] Noga Alon、Yossi Azar、Gerhard J. Woeginger 和 Tal Yadid。调度的近似方案。载于第8届年度 ACM-SIAM 离散算法研讨会论文集，493–500页，1997年。1[3]Sanjeev Arora、Elad Hazan 和 Satyen Kale。乘法权重更新方法：一种元算法及其应用。《计算理论》，8(1):121–164，2012年。4[4] Sepehr Assadi。多遍流式集合覆盖问题中紧致的空间-近似权衡。载于第36届 ACM SIGMOD-SIGACT-SIGAI 数据库系统原理研讨会论文集，321–335页，2017年。doi:10.1145/3034786.3056116。1[5] Sepehr Assadi 和 Soheil Behnezhad。随机顺序流式匹配中突破三分之二界。载于第48届年度自动机、语言与编程国际研讨会论文集，第198卷，19:1–19:13页，2021年。1[6] Sepehr Assadi、Aaron Bernstein 和 Zachary Langley。分布式负载均衡的改进界。载于第34届国际分布式计算研讨会（DISC）论文集，LIPIcs第179卷，1:1–1:15页，2020年。1[7] Sepehr Assadi、Arun Jambulapati、Yujia Jin、Aaron Sidford 和 Kevin Tian。更少遍数且空间最优的半流式二分图匹配。载于第33届年度 ACM-SIAM 离散算法研讨会论文集，627–669页，2022年。1，2

[8] Sepehr Assadi、S. Cliff Liu 和 Robert E. Tarjan。流式和大规模并行计算模型中的二分图匹配拍卖算法。载于第4届算法简洁性研讨会，SOSA 2021，线上会议，2021年1月11日至12日，165–171页，2021年。1[9] Sepehr Assadi 和 Ran Raz。双遍图流算法的近二次下界。载于第61届年度 IEEE计算机科学基础研讨会论文集，342–353页，2020年。2[10]Yossi Azar、Leah Epstein、Yossi Richter 和 Gerhard J. Woeginger。全范数近似算法。《算法杂志》，52(2):120–133，2004年。doi:10.1016/j.jalgor.2004.02.003。1，3，9[11] Leonid Barenboim 和 Gal Oren。一轮分布式备份放置及其在最大匹配和自稳定中的应用。载于第3届算法简洁性研讨会论文集，99–105页，2020年。doi:10.1137/1.9781611976014.14。1[12]MohammadHossein Bateni、Hossein Esfandiari 和 Vahab S. Mirrokni。覆盖问题的近乎最优流式算法。载于第29届 ACM算法与体系结构并行性研讨会论文集，13–23页，2017年。doi:10.1145/3087556.3087585。1[13] Aaron Bernstein。随机顺序数据流中的匹配改进界。载于第47届国际自动机、语言与编程研讨会，12:1–12:13页，2020年。1[14] Aaron Bernstein、Aditi Dudeja 和 Zachary Langley。加权图动态匹配的框架。载于第53届年度 ACM计算理论研讨会论文集，668–681页，2021年。1[15] Aaron Bernstein、Tsvi Kopelowitz、Seth Pettie、Ely Porat 和 Clifford Stein。通过重新分配，同时对每个 p-范数进行负载均衡。载于第8届理论计算机科学创新会议论文集，LIPIcs第67卷，《莱布尼茨国际信息学论文集》，第51号文章，1–4页，2017年。doi:10.4230/LIPIcs.ITCS.2017.51。1[16] Sayan Bhattacharya、Peter Kiss 和 Thatchaphol Saranurak。通过乘法权重更新实现打包-覆盖线性规划的动态算法。CoRR，abs/2207.07519，2022年。1，2[17] J. Bruno、E. G. Coffman, Jr. 和 R. Sethi。调度独立任务以减少平均完成时间。《ACM通讯》，17:382–387，1974年。doi:10.1145/361011.361064。1[18] Deeparnab Chakrabarty 和 Chaitanya Swamy。更简单、更优的最小范数负载均衡算法。载于第27届年度欧洲算法研讨会论文集，LIPIcs第144卷，《莱布尼茨国际信息学论文集》，第27号文章，1–2页，2019年。doi:10.4230/LIPIcs.ESA.2019.27。1[19] Lijie Chen、Gillat Kol、Dmitry Paramonov、Raghuvansh R. Saxena、Zhao Song 和 Huacheng Yu。可达性的近乎最优超常数遍流算法下界。载于第53届年度 ACM计算理论研讨会论文集，570–583页，2021年。2

[20] Michael Crouch 和 Daniel M. Stubbs。通过非加权匹配改进加权匹配的流式算法。载于《组合优化问题近似算法国际研讨会》第17届会议论文集，第28卷，第96–104页，2014年。doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96. 1, 18, 19[21] Andrzej Czygrinow、Michal Han´ckowiak、Edyta Szyma´nska 和 Wojciech Wawrzyniak。半匹配问题的分布式2-近似算法。载于《分布式计算国际研讨会》第26届会议论文集，计算机科学讲义第7611卷，第210–222页，2012年。doi:10.1007/978-3-642-33651-5_15. 1 [22] Jittat Fakcharoenphol、Bundit Laekhanukit 和 Danupon Nanongkai。更快的半匹配问题算法。《算法汇刊》，10(3)：第14条，23页，2014年。doi:10.1145/2601071. 1[23] Joan Feigenbaum、Sampath Kannan、Andrew McGregor、Siddharth Suri 和 Jian Zhang。半流式模型中的图问题。载于《自动机、语言与编程国际研讨会》第31届年会论文集，计算机科学讲义第3142卷，第531–543页，2004年。 1[24] Joan Feigenbaum、Sampath Kannan、Andrew McGregor、Siddharth Suri 和 Jian Zhang。半流式模型中的图问题。《理论计算机科学》，348(2-3)：207–216，2005年。 1[25] Manuela Fischer、Slobodan Mitrovic 和 Jara Uitto。半流式模型中使用 poly$(1/\epsilon)$ 遍数的确定性$(1+\epsilon)$近似最大匹配。载于第54届ACM计算理论研讨会年会论文集，第248–260页，2022年。 1[26] Ashish Goel、Michael Kapralov 和 Sanjeev Khanna。最大二分图匹配的通信复杂度与流式复杂度。载于第23届ACM-SIAM 离散算法研讨会年会论文集，第468–485页，2012年。 1[27] Venkatesan Guruswami 和 Krzysztof Onak。多遍图处理的超线性下界。Algorithmica，76(3)：654–683，2016年。 2[28]Magn´us M. Halld´orsson、Sven K¨ohler、Boaz Patt-Shamir 和 Dror Rawitz。网络中的分布式备份放置。《分布式计算》，31(2)：83–98，2018年。doi:10.1007/s00446-017-0299-x. 1[29] Nicholas J. A. Harvey、Richard E. Ladner、L´aszl´o Lov´asz 和 Tami Tamir。二分图的半匹配与负载均衡。《算法杂志》，59(1)：53–78，2006年。doi:10.1016/j.jalgor.2005.01.003. 1[30] W. A. Horn。使用并行机器最小化平均流动时间。《运筹学》，21(3)，1973年。doi:10.1287/opre.21.3.846. 1[31] Piotr Indyk、Sepideh Mahabadi、Ronitt Rubinfeld、Jonathan R. Ullman、Ali Vakilian 和 Anak Yodpinyanee。流式模型中的分数集合覆盖。载于《组合优化问题近似算法国际研讨会》第20届会议论文集，LIPIcs第81卷，第12:1–12:20页，2017年。doi:10.4230/LIPIcs.APPROX-RANDOM.2017.12. 2[32] Michael Kapralov。流式模型中匹配问题的更优界。载于第24届ACM-SIAM 离散算法研讨会年会论文集，第1679–1697页，2013年。 1

[33] Michael Kapralov。边到达模型中近似最大匹配的空间下界。载于第32届ACM-SIAM 离散算法研讨会年会论文集，第1874–1893页，2021年。 1[34] Christian Konrad、Fr´ed´eric Magniez 和 Claire Mathieu。少遍数半流式处理中的最大匹配。载于《组合优化问题近似算法国际研讨会》第15届会议论文集，第231–242页，2012年。 1[35]Christian Konrad 和 Adi Ros´en。流式处理与双方通信中的半匹配近似。《算法汇刊》，12(3)：第32条，21页，2016年。doi:10.1145/2898960. 1, 2[36]Anshul Kothari、Subhash Suri、Csaba D. T´oth 和 Yunhong Zhou。拥塞博弈、负载均衡与无政府状态代价。载于《网络的组合与算法方面》第1届会议论文集，计算机科学讲义第3405卷，第13–27页，2004年。doi:10.1007/11527954_3. 1[37]Silvio Lattanzi、Benjamin Moseley、Siddharth Suri 和 Sergei Vassilvitskii。过滤：一种在MapReduce中解决图问题的方法。载于第23届ACM算法与体系结构并行性研讨会年会论文集，第85–94页，2011年。doi:10.1145/1989493.1989505. 1[38] Jan Karel Lenstra、David B. Shmoys 和 ´Eva Tardos。无关并行机调度的近似算法。载于第 28 届 IEEE 计算机科学基础研讨会年会论文集，第217–224页，1987年。doi:10.1109/SFCS.1987.8. 2, 5, 9[39]Yixun Lin 和 Wenhua Li。单位长度机器相关作业的并行机调度。《欧洲运筹学杂志》，156(1)：261–266，2004年。doi:10.1016/S0377-2217(02)00914-1. 1[40]Renita Machado 和 Sirin Tekinay。无线传感器网络中博弈论方法综述。《计算机网络》，52(16)：3047–3061，2008年。doi:10.1016/j.gaceta.2008.07.003. 1[41] Andrew McGregor。在数据流中寻找图匹配。载于《组合优化问题近似算法国际研讨会》第8届会议论文集，第170–181页，2005年。 1[42] Andrew McGregor 和 Hoa T. Vu。最大覆盖问题的更优流式算法。载于第20届数据库理论国际会议论文集，LIPIcs第68卷，第22:1–22:18页，2017年。doi:10.4230/LIPIcs.ICDT.2017.22. 1[43] Gal Oren、Leonid Barenboim 和 Harel Levin。过载无线传感器网络中的分布式容错备份放置。载于第9届宽带通信、网络与系统国际会议论文集，第212–224页，2018年。doi:10.1007/978-3-030-05195-2_21. 1[44] Narayanan Sadagopan、Mitali Singh 和 Bhaskar Krishnamachari。基于效用的去中心化传感器网络设计。《移动网络与应用》，11(3)：341–350，2006年。doi:10.1007/s11036-006-5187-8. 1

[45] Thatchaphol Saranurak。私人通信，2022年9月。1[46]Subhash Suri、Csaba D. T´oth 和 Yunhong Zhou。自私负载均衡与原子拥塞博弈。*Algorithmica*，47(1):79–96，2004年。doi:10.1007/s00453-006-1211-4。1[47] Subhash Suri、Csaba D. T´oth 和 Yunhong Zhou。点对点系统中的非协调负载均衡与拥塞博弈。载于第3届国际点对点系统研讨会，第123–130页，2004年。doi:10.1007/978-3-540-30183-7_12。1