MATCHA：一种加速分布式优化的基于匹配的链路调度策略的复现与改进

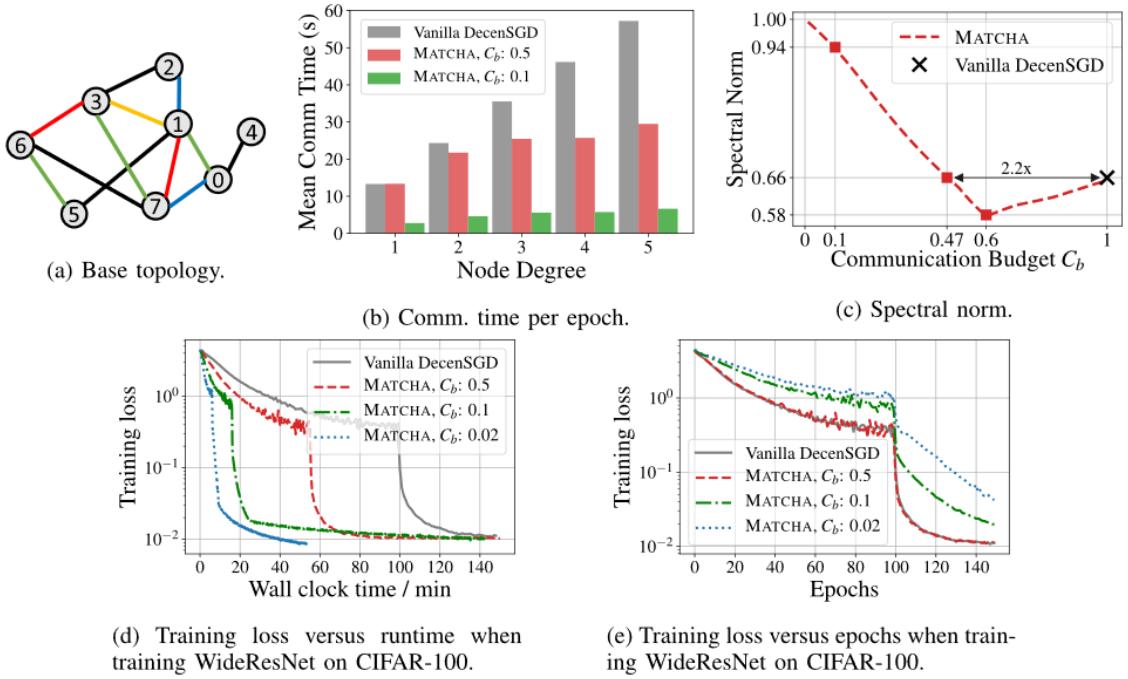
摘要

在这篇论文中，研究了在由轻量级计算节点组成的任意网络中进行分布式优化的问题，其中每个节点只能与其直接邻居进行信息的发送/接收。去中心化随机梯度下降（SGD）已被证明是在这种情况下训练机器学习模型的有效方法。尽管分布式 SGD 已经得到了广泛研究，但大多数先前的工作侧重于误差与迭代之间的收敛，而没有考虑拓扑结构如何影响每次迭代的通信延迟。例如，更密集（更稀疏）的网络拓扑会导致误差在迭代方面更快（更慢）地收敛，但会在每次迭代中产生更多（更少）的通信时间。我们提出了 MATCHA，一种可以在任意网络拓扑下实现误差和运行时权衡的算法。MATCHA 的主要思想是通过在连接关键的链路上更频繁地进行通信，以确保快速收敛，并同时通过较少地使用其他链路来最小化每次迭代的通信延迟。它通过将拓扑结构分解为匹配，并优化每次迭代中激活的匹配集来实现这种平衡。在一系列数据集和深度神经网络上的实验证实了理论分析，并表明与普通的分布式 SGD 相比，MATCHA 达到相同的训练损失所需的时间最多减少了 5 倍。MATCHA 的思想可以应用于在图中涉及与邻居进行通信步骤的任何去中心化算法。

关键词：去中心化随机梯度下降；分布式训练；通信高效方法

1 引言

大部分的有监督的机器学习问题使用经验风险最小化框架 [4], [26] 来解决，其中目标是最小化经验风险目标函数 $F(\mathbf{x}) = \sum_{s\in\mathcal{D}} \ell(\mathbf{x}, s)/|\mathcal{D}|$。优化 F（x）最常用的算法是随机梯度下降（SGD）。经典的随机梯度下降（SGD）是设计用于单个计算节点的，其关于迭代次数的误差收敛性已经得到广泛分析，并通过加速的 SGD 方法进一步改进。由于当前使用的大规模训练数据集和神经网络结构，设计分布式 SGD 实现变得势在必行，其中梯度计算和聚合分布在多个工作节点上。一种标准的梯度计算并行化方法是参数服务器框架 [5]，然而，在带宽有限的计算环境中，参数服务器和工作节点之间的持续通信成本可能非常昂贵和缓慢 [11]。提高通信效率的一种简单方法是使用本地更新或定期平均 SGD [28]，[35]，这是联邦学习 [22]，[14]，[18] 新兴领域的核心。除了通过本地更新 SGD 实现时间上的通信减少外，还可以通过消除中央聚合服务器，在分布式稀疏节点拓扑中执行训练，其中节点只能与其相邻节点通信，从而实现空间上的通信减少。在基本的分布式 SGD [24]，[19] 中，每个节点在与其邻居交换模型更新之前只进行一次本地 SGD 更新。仅在少数研究中考虑了 $\tau > 1$ 的本地更新情况，例如 [35]，[32] 它们提出并分析了周期性分布式 SGD 算法，结合了时间和空间通信的减少。



(a) Base topology.

(b) Comm. time per epoch.

(c) Spectral norm.

(d) Training loss versus runtime when training WideResNet on CIFAR-100.

(e) Training loss versus epochs when training WideResNet on CIFAR-100.

图 1. (a) 用于将 MATCHA 与普通的分散式随机梯度下降（DecenSGD）进行比较的基本网络拓扑结构。(b) MATCHA 更为显著地减少了高度节点（例如节点 1）的通信时间，这些节点往往具有较少的连接关键链路。(c) 较低的谱范数导致更好的收敛速度。通信预算 C_b 表示网络中链接上通信的平均频率。(d, e) 在 CIFAR-100 上训练 WideResNet 时的损失与时间和损失与时代曲线。

需要考虑误差与运行时间的收敛性：尽管去中心化随机梯度下降在分布式优化领域得到了广泛研究，大多数关于其收敛性的工作 [35]，[24]，[19]，[6]，[38]，[39]，[31]，[12]，[25]，[13] 仅关注达到目标误差所需的迭代或通信轮次。它们并没有明确考虑或展示拓扑结构如何影响训练运行时间，即完成每次迭代所需的挂钟时间。然而，根据算法使用的通信和聚合机制，每次迭代的运行时间与误差与迭代之间的收敛存在根本的权衡。例如，密集的网络通常能够更快地实现误差收敛，但它们会产生较高的每次迭代通信延迟。因此，为了实现最快的误差与挂钟时间的收敛，关键在于通过并置优化和调度技术共同优化迭代复杂性以及每次迭代的运行时间。仅有少数先前的研究，如 [7]，[34]，[10]，从理论的角度研究了误差与挂钟时间的收敛，但仅限于参数服务器模型。

在去中心化随机梯度下降（Decentralized SGD）设置中，我们采用这种系统感知的方法，并致力于设计节点间通信策略，以实现最快的误差与运行时间的收敛。不同的是，所有链路不应被等同对待：近期的研究提出了各种方法来减少在执行去中心化 SGD 时的通信延迟。其中一种简单的方式是执行定期的去中心化 SGD，也称为周期性去中心化 SGD（P-DecenSGD）[35]，[32]。在 P-DecenSGD 中，每个节点在与邻居同步之前进行 $\tau > 1$ 次本地模型更新。因此，在每个 τ 次迭代之后，基本拓扑中的所有链接同时被激活。其他方法包括梯度量化 [17]、异步模型聚合 [20]、随机梯度推送 [1] 和成对的流言 [6]，[3]，[21]。所有这些方法都旨在提高去中心化 SGD 方法的通信效率，但它们有一个共同的缺点，即它们对于每个节点间链接对整个图的连接性贡献一无所知。然而，每个链接对误差收敛和每次迭代的墙钟时间都有不同的影响。例如，在图 1(a) 所示的拓扑中，0–4 链接对于保持图的连接性以及确保快速误差收敛

至关重要，而删除或很少使用 1–3 链接对平均图连接性几乎没有影响，但可以帮助减少通信延迟。这使我们得出一个新的见解，即像 0–4 这样的关键连接应该更频繁地被激活。

主要贡献：MATCHA 利用这一新颖的见解，开发了一种基于如何影响拓扑的预期连通性和每次迭代的延迟的原则性方法，以优化每个链路的通信频率。与现有的通信高效 SGD 方法相比，这些方法在减少每次迭代的通信延迟时必须在误差与迭代次数的收敛速度上做出妥协，MATCHA 能够在保持（甚至提高）误差与迭代次数收敛速度的同时，灵活地减少每次迭代的通信延迟。据我们所知，这是第一项尝试在去中心化 SGD 中为任意网络拓扑实现最佳误差-运行时权衡的工作。本文的主要贡献总结如下。

1) 通过使用二分图匹配来节省通信时间：在分布式随机梯度下降的模型同步步骤中，我们在下面的第二节中证明，通信延迟是最大节点度的增函数。为了在不损害收敛性的情况下减少这种延迟，我们将图分解成匹配。每个匹配的度为一，是一组相互不相交的链路，可以并行通信，如图 1(a) 中的彩色链路所示。优化每个匹配被激活的概率，以最大化期望拓扑的代数连通性（由图拉普拉斯矩阵的第二小特征值 2 捕捉）。这导致在包含连通性关键链路的匹配上进行更频繁的通信（确保快速的误差与迭代收敛），而在其他匹配上进行更少频繁的通信（节省每次迭代的通信时间）。

2) MATCHA 允许系统设计者设置灵活的通信预算 C_b，该预算表示与普通 DecenSGD 相比，MATCHA 的相对通信时间。当 $C_b = 1$ 时，MATCHA 变为在 [19] 中研究的普通分布式 SGD。当我们设置 $C_b < 1$ 时，MATCHA 会仔细减少每个链接的通信频率，具体取决于该链接在维持图的整体连通性方面的重要性。例如，观察图 1(b) 中，通过设置 $C_b = 0.1$，MATCHA 在每次迭代中实现了 $1/0.1 = 10$ 倍的预期通信时间减少。如图 1(b) 所示，对于度数较高的节点，通信减少更为显著。这种合理的通信减少的不对称性是 MATCHA 保持快速误差与迭代收敛之间关系的关键因素。

3) 相比于普通的分布式随机梯度下降（Decentralized SGD），MATCHA 在非凸目标的收敛性分析中表现出相同或更快的误差收敛速度。在第五节中，我们进行了 MATCHA 的收敛性分析，并展示了误差对　的依赖性，其中　是混合矩阵的谱范数（稍后将正式定义）。该分析表明，在适当的通信预算下，MATCHA 实现了与普通分布式 SGD 相同或更小的　—较小的　意味着更快的误差收敛。例如，从图 1(c) 中可以观察到，MATCHA 在每次迭代中的通信预算比 DecenSGD 少 2.2 倍，但具有与 DecenSGD 相同的谱范数；如果我们设置 $C_b = 0.6$，那么谱范数甚至更低。在这种情况下，与直觉相反，MATCHA 不仅减少了每次迭代的通信延迟，还实现了更快的误差随迭代次数的收敛。

4) 我们评估了 MATCHA 在各种深度学习任务上的性能，包括图像分类和语言建模，以及针对几种基本拓扑结构，包括 Erdős-Rényi 和几何图。实证结果一致地证实了理论分析，并显示 MATCHA 在实现与普通分布式 SGD 相同的训练精度时，可以实现最多 5.2 倍的墙钟时间（计算加通信时间）缩短，如图 1(d) 所示。此外，MATCHA 实现的测试精度与普通分布式 SGD 相比相当或更好。

5) 可扩展到其他子图和计算：尽管我们目前将拓扑结构分解为匹配，但我们的方法可以扩展到其他子图，如边缘或团。它还可以与其他方法结合使用，如梯度压缩或量化 [17], [29]，异步模型聚合 [20] 以及梯度跟踪和方差减少技术 [27], [30], [37]，正如我们在第 VII 节中

所述。此外，超越分布式 SGD，MATCHA 的基于匹配的链路调度策略，这是 MATCHA 的核心思想，可扩展到任何需要在相邻节点之间频繁同步的分布式计算或共识算法。

2 预备知识和问题表述

2.1 节点拓扑与图论基础

考虑一个由 m 个工作节点组成的网络，它们对应于顶点集 $\mathcal{V} = \{1, 2, \ldots, m\}$。连接节点（顶点）的通信链路由具有边集的任意无向图 $\mathcal{G}$ 表示 $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ 每个节点 i 只能与其邻居通信，也就是说，当且仅当符合 $(i, j) \in \mathcal{E}$ 条件时，它才能与节点 j 通信。我们使用 $\mathcal{N}_i = \{j | (j, i) \in \mathcal{E}, j \in \mathcal{V}, j \neq i\}$ 去表示节点 i 的邻居索引集，节点 i 的度定义为 $d_i = |\mathcal{N}_i|$，最大节点度由 $\Delta_{\mathcal{G}} = \max_{i \in \mathcal{V}} d_i$ 表示。我们假设图 $\mathcal{G}$ 是连通的，也就是说，对于任何一对节点，都存在一条连接它们的路径。

通信图 $\mathcal{G}(\mathcal{V}, \mathcal{E})$ 也可以用邻接矩阵来表示 $\mathbf{A} \in \mathbb{R}^{m \times m}$。图形拉普拉斯矩阵 L 的定义为 $\mathbf{L} = \mathrm{diag}(d_1, \ldots, d_m) - \mathbf{A}$。图形拉普拉斯矩阵是一个正半有限矩阵，因此其特征值可以排序并表示为 $0 = \lambda_1(\mathbf{L}) \leq \lambda_2(\mathbf{L}) \leq \cdots \leq \lambda_m(\mathbf{L})$。图 $\mathcal{G}$ 是连通图当且仅当第二小特征值 $\lambda_2(\mathbf{L})$ 严格大于 0。在谱图理论 [2], [9] 中，图的代数连通性定义如下。

定义 1（代数连通性 $\lambda_2(\mathbf{L})$）：给定一个连通图 $\mathcal{G}$ 及其相应的拉普拉斯矩阵 L，$\mathcal{G}$ 的代数连通性被定义为 L 的第二个最小特征值。$\lambda_2(\mathbf{L})$ 的值越大，意味着图形越密集。

接下来，我们定义匹配子图和匹配分解的概念，它们是本文的核心。

定义 2（匹配）：匹配是 $\mathcal{G}$ 的一个子图，其中每个顶点最多与一条边相关联。

定义 3（匹配分解）：匹配分解是 G 的一组 M 个不相交的匹配子图 $\{\mathcal{G}_j(\mathcal{V}, \mathcal{E}_j)\}_{j=1}^{M}$ 使得 $\mathcal{E} = \bigcup_{j=1}^{M} \mathcal{E}_j$ 和 $\mathcal{E}_i \bigcap \mathcal{E}_j = \varnothing, \forall i \neq j$。

匹配分解不一定是唯一的，有多项式时间边着色算法 [23] 可以将给定的图分解为匹配。匹配分解的一个很好的特性是匹配数 M 与图的最大度 $\Delta_{\mathcal{G}}$ 密切相关。特别是，[23] 中的边着色算法可以保证匹配数 M 等于 $\Delta_{\mathcal{G}}$ 或 $\Delta_{\mathcal{G}} + 1$。

2.2 去中心化随机优化初步

我们认为每个工作节点只能访问它自己的本地数据集 $\mathcal{D}_i$，其中 $i \in \{1, 2, \ldots, m\}$。我们的目标是使用这个由 m 个节点组成的网络，使用联合数据集和给定的网络拓扑来训练一个公共模型 x)。特别是，我们寻求最小化目标函数 $F(\mathbf{x})$，其定义如下：

$$F(\mathbf{x}) \triangleq \frac{1}{m} \sum_{i=1}^{m} F_i(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{|\mathcal{D}_i|} \sum_{s \in \mathcal{D}_i} \ell(\mathbf{x}; s) \tag{1}$$

其中 x 表示模型参数（例如，神经网络的权重和偏差），$F_i(\mathbf{x})$ 是局部目标函数，s 表示单个数据样本，并且 $\ell(\mathbf{x}; s)$）是由学习模型定义的样本 s 的损失函数。

基本分散式随机梯度下降更新规则：为了最小化目标函数 (1)，从 [24] 和 [33] 的开创性工作开始，人们开发了许多去中心化优化算法。在典型的分散优化算法中，每个工作节点交替进行本地计算和与邻近节点通信。例如，在基本分散式随机梯度下降（DecenSGD）[19]、[38]、[13] 中，所有节点都会迭代运行以下程序：

1) 并行局部计算：节点 i 根据模型参数的局部版本计算随机梯度：$g_i(\mathbf{x}_i^{(k)}) = \frac{1}{|\mathcal{B}|}\sum_{\boldsymbol{s}\in\mathcal{B}} \nabla\ell(\mathbf{x}_i^{(k)}; \boldsymbol{s})$ 其中 $\mathcal{B} \subseteq \mathcal{D}_i$ 是从本地数据集中随机采样的小批量 $\mathcal{B}$ 然后，节点 i 使用随机梯度更新其局部参数：$\mathrm{x}_i^{(k+\frac{1}{2})} = \mathrm{x}_i^{(k)} - \eta g_i(\mathrm{x}_i^{(k)})$ 其中 η 表示学习速率。

2) 与邻居交流：为了达成共识，工作节点需要与邻居交换模型参数。这个过程涉及双向通信。

   - 节点 i 将其局部模型参数 $\mathrm{x}_i^{(k+\frac{1}{2})}$ 发送到其相邻节点 $\mathcal{N}_i$；
   - 节点 i 从它的每个邻居接收模型参数。在这一步之后，节点 i 应该有所有邻居的模型参数 $\left\{\mathbf{x}_j^{(k+\frac{1}{2})}\right\}_{j\in\mathcal{N}_i}$。

3) 将本地模型与邻近模型混合：由于节点希望共同训练一个共同的模型 x，因此它们会执行一个共识步骤，将自己的模型 $\mathrm{x}_i^{(k+\frac{1}{2})}$ 与邻近的模型进行加权平均。因此，节点 i 将其模型更新为：$\mathrm{x}_i^{(k+1)} = \sum_{j\in\mathcal{N}_i\cup\{i\}} W_{ji}\mathrm{x}_j^{(k+\frac{1}{2})}$，其中 W_{ji} 是混合矩阵 $\mathbf{W} \in \mathbb{R}^{m\times m}$ 的第（j，i）个元素。它表示更新节点 i 的模型时分配给邻居 j 的权重。观察 $W_{ij} \neq 0$ 当且仅当节点 i 和节点 j 相连，即 $(i, j) \in \mathcal{E}$。

综合以上步骤，我们可以写出基本分散式随机梯度下降的更新规则如下：

$$\mathrm{x}_i^{(k+1)} = \underbrace{\sum_{j=1}^{m} W_{ij}}_{\text{consensus step}} \underbrace{\left[\mathrm{x}_j^{(k)} - \eta g_j\left(\mathrm{x}_j^{(k)}\right)\right]}_{\text{parallel local computation}} \tag{2}$$

为了在（2）中的节点之间实施一致性，通常的做法是使用对称和双随机（即，每列/行的和为 1）混合矩阵 W 。通常的选择是等权矩阵，定义如下：

$$\mathrm{W} = \mathrm{I} - \alpha\mathrm{L} \tag{3}$$

其中 是可调参数，L 表示图 $\mathcal{G}$ 的拉普拉斯矩阵。定义（3）通过构造使 W 对称和双随机，并在文献 [6]、[16] 中广泛使用。 越大，相邻模型的权重越高。例如，如果节点 1 只连接到节点 2 和节点 3，则图拉普拉斯 L 的第一行是 [2，−1，−1，0，...，0]，对应的第一行 W 是 [1−2，，，，0，...，0]。虽然我们在本节中使用固定的 W 来描述分散的 SGD，但它也可以随着迭代而变化。

分散 SGD 的收敛性分析：当目标函数（1）是非凸函数时，那么收敛性的典型度量是梯度范数 $\|\nabla F(\mathbf{x})\|$。特别地，我们有下面的非正式引理。

引理 1（DecenSGD 的收敛保证，[35]，[19]）：假设 $\bar{\mathrm{x}}$ 表示所有工作节点的平均模型，并且节点处的所有局部模型都从同一点 $\bar{\mathrm{x}}(1)$ 初始化。然后在随机梯度的标准假设下（正式说明见第五节），我们在 K 次迭代后：

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\|\nabla F\left(\bar{\mathbf{x}}^{(k)}\right)\|^2 = \mathcal{O}\left(\frac{1}{\sqrt{mK}}\right) + \mathcal{O}\left(\frac{m}{K}\frac{\rho}{(1-\sqrt{\rho})^2}\right) \tag{4}$$

其中 K 是总迭代次数， 表示矩阵 $\mathbf{W}^2 - \mathbf{1}\mathbf{1}^\top/m$ 的谱范数（即最大奇异值）。

引理 1 表明，DecenSGD 的收敛速度受谱范数 $\rho = \|\mathbf{W}^2 - \mathbf{1}\mathbf{1}^\top/m\|_2 < 1$ 控制。 越小，收敛误差界越好。通常， 的值与网络拓扑的连通性有关。当工作节点可以与任何其他节点通信

时（即网络拓扑完全连接），则 =0，并且（4）中的第二项达到其最小值。当网络拓扑非常稀疏时， 将接近 1，使得（4）中的第二项接近无穷大。

注 1（$\lambda_2(\mathbf{L})$ 越大，误差收敛越快）：当混合矩阵的形式为 $\mathbf{W}=\mathbf{I}-\alpha\mathbf{L}$ 时，则（详细推导见 [15]）

$$\rho=\max\{(1-\alpha\lambda_m(\mathbf{L}))^2,(1-\alpha\lambda_2(\mathbf{L}))^2\} \tag{5}$$

$$\geq\frac{1-\frac{\lambda_2(\mathbf{L})}{\lambda_m(\mathbf{L})}}{1+\frac{\lambda_2(\mathbf{L})}{\lambda_m(\mathbf{L})}} \tag{6}$$

其中通过设置 $\alpha=2/(\lambda_2(\mathbf{L})+\lambda_m(\mathbf{L}))$ 来实现相等。因此，$\lambda_2(\mathbf{L})/\lambda_m(\mathbf{L})$[1] 较大的图将具有较小的 值，因此具有较好的收敛误差界。$\lambda_2(\mathbf{L})/\lambda_m(\mathbf{L})$ 的值对 $\lambda_2(\mathbf{L})$）更敏感，因为 $\lambda_m(\mathbf{L})$ 总是被远离零的平均节点度下界 [15]。因此，较大的 $\lambda_2(\mathbf{L})$ 将导致较好的误差收敛界。

我们在 MATCHA 中使用上述观察，并设计一个拓扑序列，使期望图的 $\lambda_2(\mathbf{L})$ 最大化，然后优化 的值，以最小化相应的谱范数 。所得到的混合矩阵序列减少了每次迭代的通信延迟，同时保持或改善了误差收敛界限。

2.3 了解网络拓扑如何影响每次迭代的运行时间

观察引理 1 给出了分散 SGD 关于迭代次数 K 和谱范数 的收敛界，没有考虑每次迭代花费的时间。在本节中，我们的目标是理解网络拓扑如何影响每次迭代的运行时。运行时包括执行与相邻节点的模型双向交换所需的本地计算时间和通信时间 ${T_{\mathcal{G}}}^{\text{mm}}$（上述普通 DecenSGD 更新规则的步骤 2）。由于在带宽受限的环境中，通信时间 ${T_{\mathcal{G}}}^{\text{mm}}$ 通常支配本地计算时间，因此我们重点了解 ${T_{\mathcal{G}}}^{\text{mm}}$ 如何受到网络拓扑的影响。特别是，我们有以下命题。

命题 1：分散优化算法每次迭代的预期通信时间 $\mathbb{E}[T_{\mathcal{G}}^{\text{comm}}]$ 随着图中的最大度单调增加，即，

$$\mathbb{E}[T_{\mathcal{G}}^{\text{comm}}]\geq t(\Delta_{\mathcal{G}}) \tag{7}$$

其中 $t(\cdot)$ 是单调递增函数，$\Delta_{\mathcal{G}}=\max_{i\in\mathcal{V}}d_i$，图 $\mathcal{G}$ 中的最大节点度。

在本节的其余部分，我们证明了为什么命题 1 是正确的，而不考虑链路延迟分布和节点间通信协议。如果读者相信命题 1，他们可以跳到第三节，描述提出的 MATCHA 算法。

节点 i 的每次迭代通信时间 T_i 随着其度 d_i 而增加：回想一下，分散 SGD 的通信步骤（参见 2.1 节中的步骤 2）需要模型的双向通信——每个节点需要将其模型参数发送给其邻居，然后从每个邻居接收回一个模型。在不丧失一般性的前提下，假设最快的节点在时间 0 完成其本地计算（第 2.1 节中的步骤 1）。从这个时间开始，设随机变量 $T_{i\to j}^{\text{send}}$ 分别表示节点 i 完成向节点 j 发送其模型的时间（包括等待其他并发节点间通信所花费的任何空闲时间）。因此，节点 i 需要时间

$$T_i=\max\left(\max_{j\in\mathcal{N}_i}T_{i\to j}^{\text{send}},\max_{j\in\mathcal{N}_i}T_{j\to i}^{\text{send}}\right) \tag{8}$$

在每次迭代中与其邻居进行通信。由于最慢的相邻节点总是瓶颈，$\mathbb{E}[T_i]=t(d_i)$，其中 t $(\cdot)$ 是单调递增函数。

函数 t 的确切形式取决于随机变量 $T_{i\to j}^{\text{send}}$ 的分布，而随机变量 $T_{i\to j}^{\text{send}}$ 又取决于许多因素，如节点间通信和冲突避免协议、链路延迟、后台进程等。例如，如果节点 i 顺序地将其模型发送到其 d_i 邻居，其中每次传输花费时间 S_i，则 $\max_{j\in\mathcal{N}_i}T_{i\to j}^{\text{send}}=d_iS_i$，并且 $\mathbb{E}[T_i]$ 随 d_i 线

性增加。相反，如果节点 i 在时间 $S_i \sim \mathrm{Exp}(\mu)$ 向其邻居广播其模型，即 i.i.d。跨节点，则 $\max_{j\in\mathcal{N}_i} T_{i\to j}^{\mathrm{send}} = S_i$ 和 $\mathbb{E}[T_i] = \max_{j\in N_i} S_j \approx \frac{\log(1+d_i)}{\mu}$，其在 d_i 中再次单调增加。

通信时间 $T_{\mathcal{G}}^{comm}$ 随着最大节点度的增加而增加：现在让我们用 T_i 来表示每次迭代的通信时间 $T_{\mathcal{G}}^{comm}$。在所有节点可以同时与各自的邻居通信的理想网络中，$T_{\mathcal{G}}^{\mathrm{comm}} = \max_{i\in\mathcal{V}} T_i$。然而，实际上，

$$\mathbb{E}[T_{\mathcal{G}}^{\mathrm{comm}}] \geq \mathbb{E}[\max_{i\in\mathcal{V}} T_i] \tag{9}$$

$$\geq \max_{i\in\mathcal{V}} \mathbb{E}[T_i] \tag{10}$$

$$= t(\Delta_{\mathcal{G}}) \tag{11}$$

最后一个等式（11）来自于 $t(\cdot)$ 是一个递增函数。因此，我们证明了命题 1 为真。此外，只有当所有节点可以并行执行它们的发送和接收通信时，等式才在（10）中成立。然而，在实践中实现这一点需要小心地将时间/频率资源分配给每个链路上的发送/接收通信，以避免冲突。下面，我们展示了如何编排节点间通信的示例。

- 频分复用模型（FDM）：为了避免节点间通信冲突，每个节点可以有一个专用的频率信道将其模型发送给邻居，并有与邻居发送频率相对应的 di 接收信道。在总共有 m 个频率的情况下，所有节点确实可以并行通信，因此 $\mathbb{E}[T_{\mathcal{G}}^{\mathrm{comm}}]$ 将达到（10）中的下限。或者，我们可以将拓扑分解成 M 个匹配，并为每个匹配分配一个发送和一个接收频率。此策略再次达到（10）的下限，但是 $2\Delta_{\mathcal{G}}$（通常小于 m）频率信道。

- 时分复用模型（TDM）：在时分复用中，系统中的所有节点使用相同的频率发送和接收消息。为了避免冲突，每个节点间传输都需要分配一个专用的时隙。一种简单的方法是给每个节点分配一个时隙，将其模型广播给邻居。这将导致 $\mathbb{E}[T_{\mathcal{G}}^{\mathrm{comm}}]$ 与节点数 m 线性缩放。相反，我们的链路调度算法 MATCHA 提出了一种更有效的 TDM 实现。我们将图分解为 $\mathrm{M} = \Delta_{\mathcal{G}}$ 或 $\mathrm{M} = \Delta_{\mathcal{G}} + 1$ 匹配，然后为每个匹配分配一个时隙。因为匹配包含不冲突的链接，所以匹配中的所有链接都可以在同一时隙被激活。结果，$\mathbb{E}[T_{\mathcal{G}}^{\mathrm{comm}}] \propto \Delta_{\mathcal{G}}$，它也等式地达到了（10）中的下界。

3 本文方法

在第二节中，我们证明了每次迭代的通信延迟随着最大节点度 $\Delta_{\mathcal{G}}$ 而增加。另一方面，我们也证明了一个更好的连通图（通常具有更高的 $\Delta_{\mathcal{G}}$）给出了更快的误差对迭代收敛。我们提出的链路调度策略 MATCHA 寻求这些对立力量之间的最佳权衡。MATCHA 生成时变通信拓扑，最大化预期图的代数连通性，同时保持预期最大度较小。通过这样做，我们自动在连接关键链路上更频繁地通信（保持快速错误收敛），而在其他链路上更不频繁地通信（减少每次迭代的通信延迟）。我们描述了生成时变拓扑序列的匹配分解采样（如图 2 所示）过程。本节描述的所有步骤都是预处理步骤，可以在培训开始前执行。因此，MATCHA 不会给总训练时间增加任何每次迭代的开销。

3.1 匹配分解

首先，我们将基本节点拓扑分解成总共 M 个不相交的子图，即：$\{\mathcal{G}_j(\mathcal{V},\mathcal{E}_j)\}_{j=1}^{M},\mathcal{E} = \bigcup_{j=1}^{M}\mathcal{E}_j$ 和 $\mathcal{E}_i\bigcap\mathcal{E}_j = \varnothing, \forall i \neq j$，如第二节所定义。在每次训练迭代中，只有这些子图的子集被激活。在命题 1 中已经证明了通信延迟是最大节点度的单调递增函数。因此，为了节省通信延迟，激活的拓扑具有比基本节点拓扑更小的最大度是特别感兴趣的。因此，这里的一个关键设计问题是：使用哪种基本拓扑的分解 $\{\mathcal{G}_j\}$，以便我们可以轻松灵活地控制激活拓扑的最大程度?

在本文中，我们选择使用匹配作为分解基础。匹配中的所有节点最多具有 1 度。节点间链路是不相交的，可以并行操作。如第二节所示，匹配分解的一个很好的性质是，使用边缘着色算法 [38]，可以证明保证匹配的数量等于 $\Delta_{\mathcal{G}}$ 或 $\Delta_{\mathcal{G}}+1$，其中 $\Delta_{\mathcal{G}}$ 是最大节点度。对于几乎所有的图，在 [8] 中已经证明了存在一个分解方案，使得 $\mathrm{M} = \Delta_{\mathcal{G}}$。也就是说，具有最高度的节点出现在所有 M 个匹配中。因此，如果在某个迭代中有 N 个匹配被激活用于通信，那么被激活拓扑的最大度正好是 N。

使用匹配分解的另一个关键好处是，它可以自动构建链路调度方案，该方案可以实现每次迭代的通信时间下限（7）。如第 II-C 节所述，在 TDM 通信模型中，可以为每个匹配分配一个时隙，并顺序通信它们。在 FDM 通信模型中，可以为每个匹配分配一个频率，并并行通信。因此，通过使用匹配，可以通过调整 N（在每次迭代中要激活的匹配数）来容易地控制通信延迟 $\mathbb{E}[T_{\mathcal{G}}^{\mathrm{comm}}]$=t（$\Delta_{\mathcal{G}}$）= t（N）。

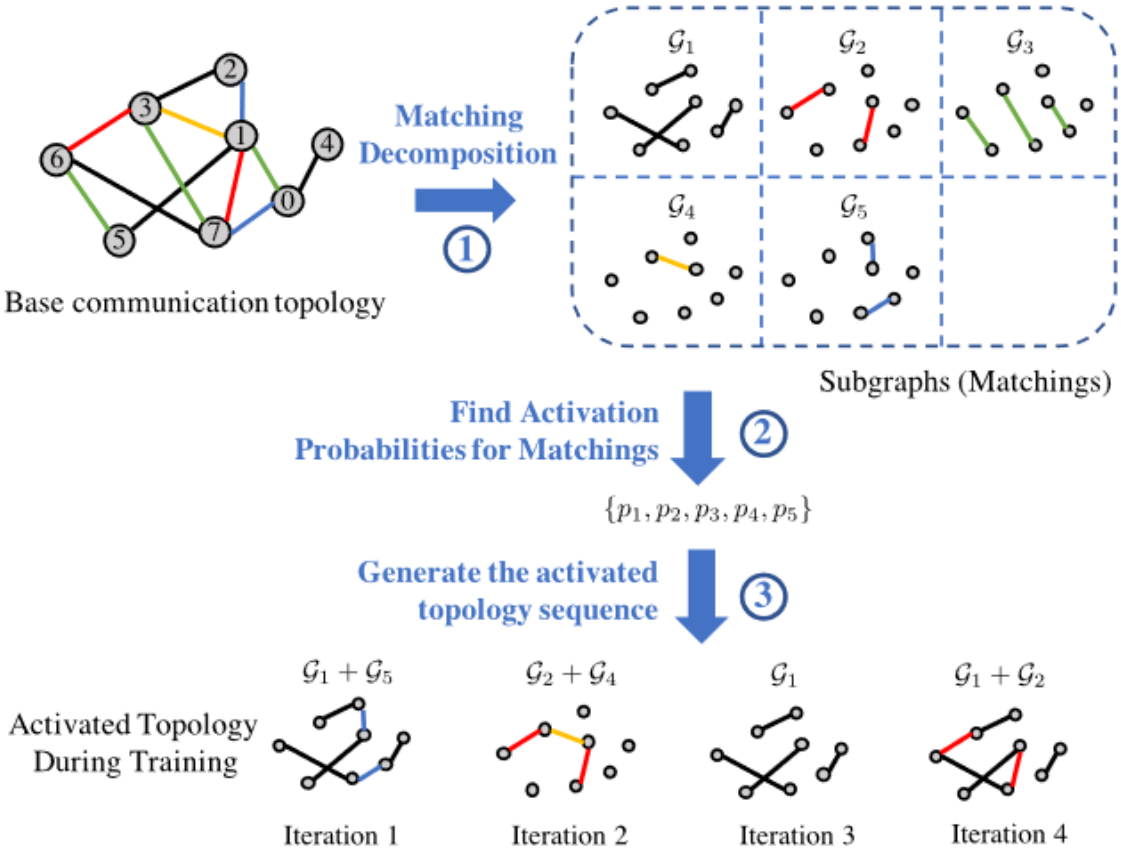


图 2. MATCHA 插图。我们将基本通信图分解成不相交的子图（特别是匹配，以便并行化通信）。在每一轮通信中，我们激活这些匹配的子集来构建基本拓扑的稀疏子图。工作节点仅通过激活的拓扑进行同步。如果一个节点不涉及激活的子图，它只执行一次本地更新。

3.2 计算匹配激活概率

为了激活 M 个匹配的子集，在每次训练迭代中，我们为每个匹配 $\mathcal{G}_j$ 分配一个独立的伯努利随机变量 B_j，其概率为 p_j 为 1，否则为 0，$\forall j \in \{1,\ldots,M\}$。仅当 B_j 的实现为 1 时，匹配 j 中的链接才会用于共识步骤中的信息交换。使用这种随机采样方案，可以将每次迭代的预期通信时间写为：

$$\text{Expected Comm. Time} = \mathbb{E}\left[t(\Delta_{\mathcal{G}(k)})\right] = \mathbb{E}\left[t\left(\sum_{j=1}^{M}\mathcal{B}_j\right)\right] \tag{12}$$

其中 $\mathcal{G}^{(k)}$ 表示在第 k 次迭代时激活的拓扑。我们进一步定义 $\mathbb{E}[B_j] = p_j$ 为 j 次匹配的激活概率。当所有 p_j 都等于 1 时，该算法简化为普通的 DecenSGD，完成一个一致性步骤需要 t（M）个时间。为了减小通信时延，我们定义了通信预算 C_b>0，并施加了约束 $\mathbb{E}\left[t\left(\sum_{j=1}^{M}\mathcal{B}_j\right)\right] \leq C_b \cdot t(M)$。例如，$C_b$=0.1 意味着抹茶每次迭代的预期通信时间是在去中心化随机梯度下降每次迭代时间的百分之十。方程（12）可以在 TDM 模型中进一步简化。当我们忽略所有常数时，可以得出 $\mathbb{E}\left[t\left(\sum_{j=1}^{M}\mathcal{B}_j\right)\right] = \mathbb{E}\left[\sum_{j=1}^{M}\mathcal{B}_j\right] = \sum_{j=1}^{M} p_j$。

如前所述，MATCHA 的关键思想是更加重视关键环节。这是通过选择一组激活概率来实现的，该激活概率在给定通信时间约束的情况下最大化预期图的连通性。也就是说，我们解决了优化问题：

$$\begin{aligned} \max_{p_1,\ldots,p_M} \quad & \lambda_2\left(\sum_{j=1}^{M} p_j \mathrm{L}_j\right) \\ \text{subject to} \quad & \mathbb{E}\left[t\left(\sum_{j=1}^{M}\mathrm{B}_j\right)\right] \leq C_b \cdot t(M), \\ & 0 \leq p_j \leq 1, \forall j \in \{1,2,\ldots,M\}, \end{aligned} \tag{13}$$

其中 L_j 表示第 j 个子图的拉普拉斯矩阵，并且 $\sum_{j=1}^{M} p_j \mathbf{L}_j$ 可以被认为是期望图的拉普拉斯矩阵。此外，回想 λ_2 表示代数连通性，并且是凹函数 [2]，[16]。优化问题（13）可以通过数值方法来解决。在 TDM 通信模型中，（13）中的约束减少到 $\sum_{j=1}^{M} p_j \leq C_b M$。因此，（13）是一个凸问题，可以有效地求解。通常情况下，λ_2 值越大，意味着图的连通性越好，在许多分散应用中达成共识的速度越快 [16]。

3.3 生成随机拓扑序列

给定通过求解（13）获得的激活概率，在每次迭代 k 中，我们为每个匹配 $j = 1,\ldots M$ 生成独立的伯努利随机变量 $\mathcal{B}_j^{(k)}$。因此，在第 k 次迭代中，激活拓扑 $\mathcal{G}^{(k)}(\mathcal{V},\mathcal{E}^{(k)})$，其中 $\mathcal{E}^{(k)} = \bigcup_{j=1}^{M}\mathcal{B}_j^{(k)}\mathcal{E}_j$ 是稀疏的，甚至是断开的。请注意，如果一个节点没有激活的链接，那么它仍然会继续对当前的本地模型进行本地更新。除了拓扑序列之外，还可以得到相应的拉普拉斯矩阵序列：$\mathcal{L}^{(k)} = \sum_{j=1}^{M}\mathcal{B}_j^{(k)}\mathcal{L}_j, \forall k \in \{1,2,\ldots\}$。所有这些信息都可以在开始训练过程之前获得并预先分配给工人节点。实际上，服务器不需要发送整个拉普拉斯矩阵序列 $\{\mathcal{L}^{(1)},\mathcal{L}^{(2)},\ldots\mathcal{L}^{(K)}\}$ 到工作节点，其大小可以高达 MMK。相反，服务器可以首先广播匹配分解 $\{\mathbf{L}_1,\mathbf{L}_2,\ldots,\mathbf{L}_M\}$，然后将激活的索引序列发送到所有节点，其大小最多不超过 MK。此外，与训练期间传输神经网络参数的成本（神经网络的大小乘以迭代次数）相比，训练前的这种额外通信成本可以

忽略不计。此外，还有另一种简单的方法来消除网络拓扑的通信。我们可以给所有客户端分配相同的随机种子，让它们自己生成激活图序列。通过这样做，我们可以避免发送整个激活图序列。

3.4 优化拓扑序列的混合矩阵权重

为了在运行具有时变拓扑的分散 SGD 时充分利用 MATCHA 生成的激活子图序列，我们需要优化在（2）的一致性步骤中局部模型被平均在一起的比例。如第二节所述，通常的做法是使用等权重混合矩阵，如 [6]、[16]、[36]：

$$\mathbf{W}^{(k)} = \mathbf{I} - \alpha\mathcal{L}^{(k)} = \mathbf{I} - \alpha\sum_{j=1}^{M}\mathbf{B}_j^{(k)}\mathbf{L}_j, \tag{14}$$

其中 $\mathcal{L}^{(k)} = \sum_{j=1}^{M}\mathcal{B}_j^{(k)}\mathcal{L}_j$ 表示第 k 次迭代时的图拉普拉斯。每个矩阵 $\mathbf{w}(k)$ 在构造上是对称的和双随机的，参数 α 表示一致性步骤中邻居信息的权重。为了保证对于光滑和非凸损失，分散 SGD 的每次迭代后都有足够的减少，谱范数 $\rho = \|\mathbb{E}[\mathbf{W}^{(k)\top}\mathbf{W}^{(k)}] - \frac{1}{m}11^\top\|_2$ 必须小于 1。一般而言，ρ 的值越小，误差范围就越小。

在下面的定理中，我们通过公式化一个半定规划问题来优化 ，并表明对于具有任意通信预算 C_b>0 的 MATCHA，人们总是可以找到 的值，对于该值，由此产生的谱范数 <1。

定理 1：设 $\{\mathcal{L}^{(k)}\}$ 表示对于连通基图 $\mathcal{G}$，由 MATCHA 生成的具有任意通信预算 C_b>0 的拉普拉斯矩阵序列。如果混合矩阵定义为 $\mathbf{W}^{(k)} = \mathbf{I} - \alpha\mathcal{L}^{(k)}$，则存在 α 的一个值域使得 $\rho = \|\mathbb{E}[\mathbf{W}^{(k)\top}\mathbf{W}^{(k)}] - \frac{1}{m}\mathbf{1}\mathbf{1}^\top\|_2 < 1$，这保证了 MATCHA 收敛到一个驻点。

α 的值可以通过求解以下半定规划问题得到：

$$\begin{aligned} &\min_{\rho,\alpha,\beta}\rho, \\ &\text{subject to}\alpha^2 - \beta \le 0, \\ &\mathbf{I} - 2\alpha\overline{\mathbf{L}} + \beta\left[\overline{\mathbf{L}}^2 + 2\widetilde{\mathbf{L}}\right] - \frac{1}{m}11^\top \preceq \rho\mathbf{I} \end{aligned} \tag{15}$$

其中 β 为辅助变量，$\overline{\mathbf{L}} = \sum_{j=1}^{M}p_j\mathbf{L}_j$ 和 $\tilde{\mathrm{L}} = \sum_{j=1}^{M}p_j(1-p_j)\mathbf{L}_j$。

类似于优化问题（13），半定规划问题（15）只需要在训练开始时求解一次，并且这个额外的计算时间（几秒甚至更少）与总训练时间（小时）相比可以忽略不计。

请注意，MATCHA 中的激活概率也会隐式影响谱范数。理想情况下，给定通信预算，应该通过（15）这样的公式联合优化 p_i 和 α。然而，由此产生的优化问题是非凸的，无法有效解决。因此，在 MATCHA 中，我们分别优化 p_i 和参数 α。通过 (13) 优化 p_i 可以被认为是最小化谱范数 ρ 的上限。

对通信预算 C_b 的依赖性：虽然很难根据通信预算 C_b 得到 ρ 的解析形式，但在图 3 中，我们给出了通过求解 TDM 通信模型中的优化问题 (13) 和 (15) 而获得的一些数值结果。回想一下，较低的谱范数 ρ 意味着更好的迭代误差收敛。请注意，人们总能找到一个 C_b 值，该值保持了与 vanilla DecenSGD 相同的光谱范数，但仅花费 2-3 倍的通信时间。通过设置适当的预算，MATCHAcan 甚至比 vanilla DecenSGD 具有更低的光谱范数。换句话说，MATCHAachieves 在错误运行时权衡中实现了双赢与普通的 DecenSGD 相比，它将通信延迟降低了 2-3 倍，但具有相同甚至更好的收敛保证！图 3 的数值结果可以作为实际选择 C_b 时的指南。

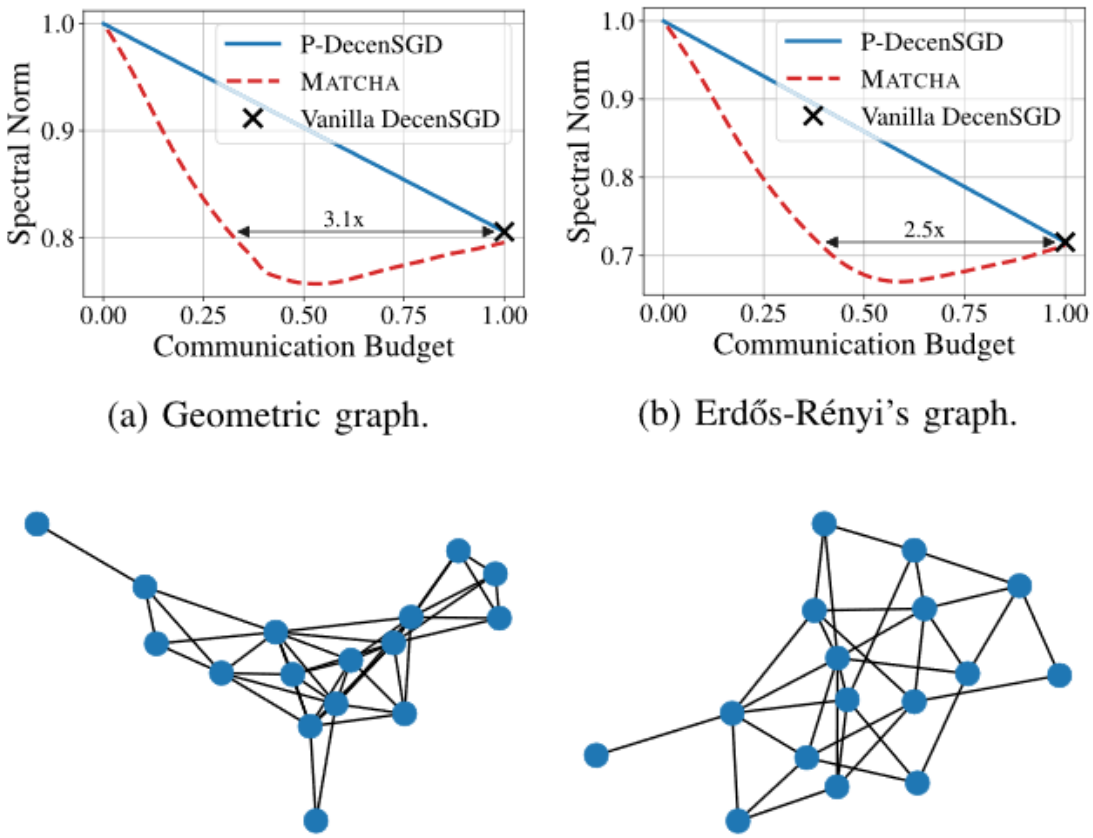


图 3. MATCHA 中频谱范数 ρ 如何随通信预算变化的示例。在（a）和（b）中，都有 16 个工作节点。MATCHA 通常比 vanilla DecenSGD 花费 2–3 倍的通信时间，同时保持完全相同甚至更低的 ρ 值（即相同或更好的错误上限）。

4 复现细节

4.1 与已有开源代码对比

我们知道一个图的通信成本与每一轮的最大节点度有关。因此，我们需要找到一种在尽可能保证图的最优连通性的同时满足预期预算的划分方法，让分布式 SGD 以最快的速度收敛。原论文中基于贪婪策略的边缘着色算法是一种很好的方法，可以保证获得的子图数量不超过最大节点度。此外，这种划分方法允许我们方便地控制通信延迟（确保在任何给定时间同时激活的子图的最大数量），可以通过调整激活概率来最大化期望中矩阵的第二小特征值。这进一步增加了谱范数 ρ 的大小，从而加速了训练的收敛速度，稀疏图中贪婪策略的时间复杂度约为 $O(VlogV + E)$。然而，我们在贪婪策略下发现的匹配分解可能远非最优。

我们可以尝试识别所有满足标准的匹配分解策略，并在预处理过程中计算它们的最优连接值，以实现理论上的最优分解方法。下面是我提出的改进方法：基于回溯的网络拓扑匹配分解（BBNTMD）算法。该算法用于尝试寻找所有可能性。图 4 中显示了 BBNTMD 的伪代码。

为了更好地理解该算法，我们使用图 5 的算法执行图进行解释。该算法与原始核心思想的区别在于它融合了回溯和贪婪策略，基于边缘在状态之间转换。在这种表示中，子节点的空圆圈表示将当前节点的特定边添加到匹配中。三角形表示从图形中提取匹配项。实心圆表

示当前节点没有要添加到正在进行的匹配中的边。最后，五边形象征着找到一种方法将图形分解成匹配。

尽管看起来有 2^E 的时间复杂度，但在稀疏图中，它远没有那么高。实际上，复杂度是 $O(\text{num} \cdot (v \log v))$，其中“num”表示图可以分解成的可能匹配的计数。在稀疏图中，这种成本在预处理期间几乎是可以忽略不计的。

Algorithm 1 BBNTMD: Finding all ways to divide a graph into matching

Require: Graph $G(V, E)$, integer *position*, set *nodeSet*, match *match*

```
Form position to end, find a node1 in nodeSet
if can't find node1 then
    if nothing in nodeSet then
        Add match to graphs and add graphs to allWay
    else
        Add match to graphs
        Sort the rest of Graph(V, E)
        BBNTMD()
    end if
else
    From node1 to end, find a node2 in nodeSet such that
    node2 is linked to node1
    Remove edge(node1, node2)
    BBNTMD()
    Add edge(node1, node2)
    From node2 to ends, find node3 with degree equal to
    node2's degree
    Remove edge(node1, node3)
    BBNTMD()
    Add edge(node1, node2)
    if can't find node2 then
        BBNTMD()
    end if
end if
```

图 4. BBNTMD 的伪代码

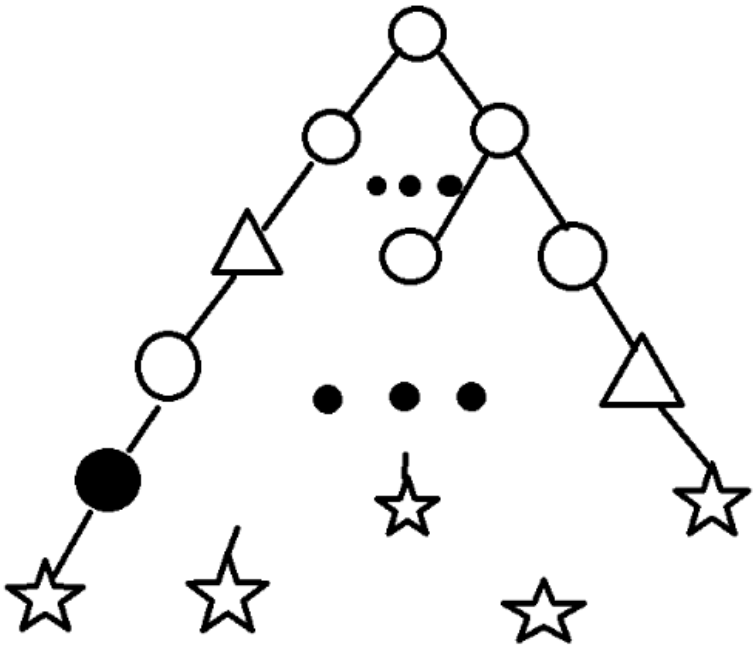


图 5. 算法执行图

4.2 实验环境搭建

Python 3.6.9
Pytorch 1.0.0
Torchvision 0.2.1
Openmpi 3.0.0
Mpi4py 3.1.5
Cvxpy 1.1.24

4.3 界面分析与使用说明

直接运行代码就可以

4.4 创新点

提出了基于回溯的网络拓扑匹配分解（BBNTMD）算法，该算法融合了回溯和贪婪策略，基于边缘在状态之间转换。通过识别所有可能的匹配分解策略，并在预处理过程中计算它们的最优连接值，实现了理论上的最优分解方法。该算法在稀疏图中具有相对较低的时间复杂度，为寻找最优解提供了高效的方法。

5 实验结果分析

图 6，8，10 展示了原文算法在 CIFAR-10 数据集上的训练效果，而图 7，9，11 则呈现了采用改进匹配分解的训练效果。通过对比两者，我们可以观察到在 CIFAR-10 上采用改进匹配分解的训练方式所带来的显著性提升。具体而言，训练速度相较于原文算法提升了约 6.66%。

这一改进的训练方法在 CIFAR-10 数据集上的表现差异清晰可见，突显了采用基于回溯的网络拓扑匹配分解（BBNTMD）算法的优势。新方法不仅有效应用了贪婪策略的边缘着色算法，而且通过改进的匹配分解策略进一步提高了训练效率。这样的优化在实际应用中将为分布式 SGD 等任务提供更为高效的图划分方式，从而优化通信成本并加速整个训练过程。这次训练效果的比较进一步验证了算法改进的实用性和性能提升。

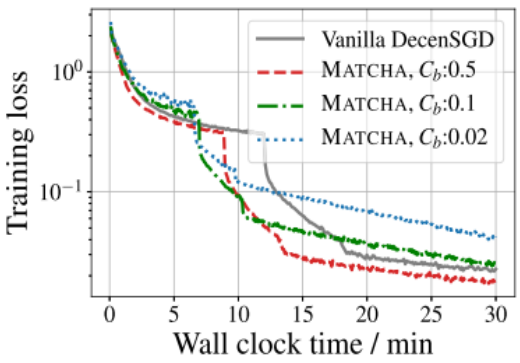


图 6. 损失与运行时间原论文

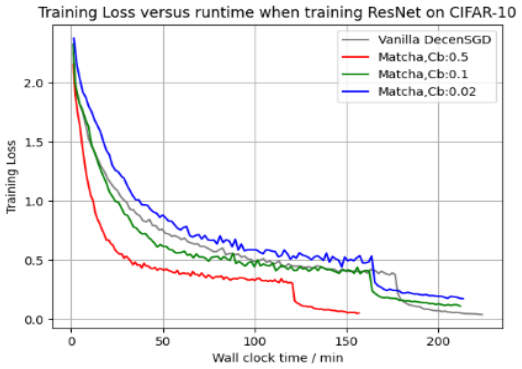


图 7. 改进后

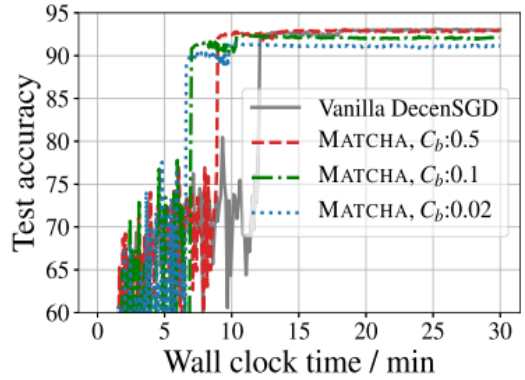


图 8. 准确率与运行时间原论文

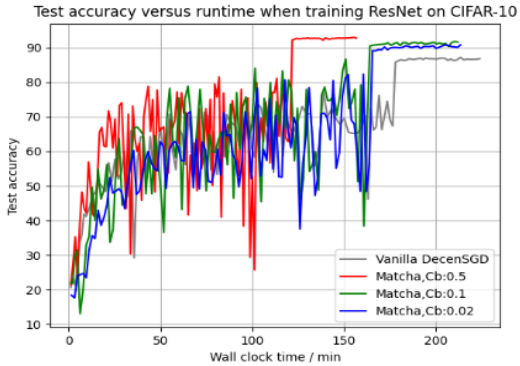


图 9. 改进后

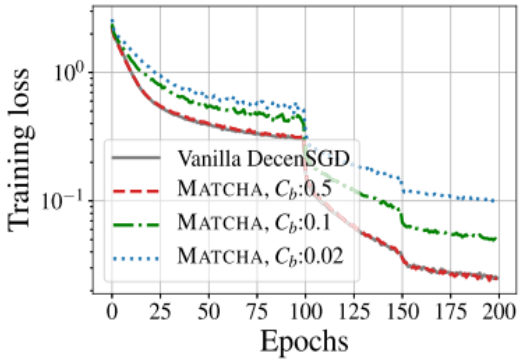


图 10. 损失与运行轮次原论文

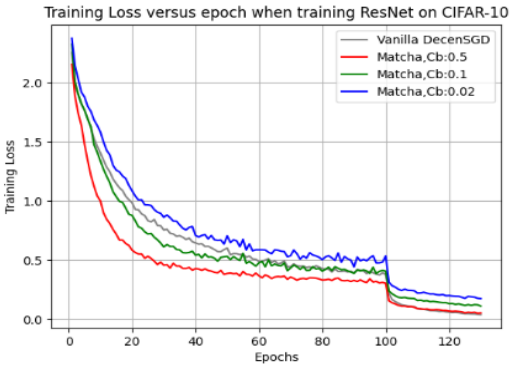


图 11. 改进后

6 总结与展望

我们提出的 BBNTMD 算法具备寻找全局最优解的能力，然而需要特别注意的是其算法成本。尽管该算法在我们的测试集上能够运行，但在图变得密集时，额外的计算开销可能变得难以接受。然而，在一般的应用场景中，该算法依然表现出可接受且有效的特性。为了拓

展算法的适用范围，我们迫切需要寻找更好的剪枝方法，或者考虑使用其他不需要回溯的策略来近似最优解。

这样的优化探索将使得我们的算法更加灵活，能够在更广泛的应用环境中发挥其优越性能。在处理图密度增加的情况下，寻找更为高效的剪枝机制将是关键，以确保算法在各种实际应用中都能够保持高效性。这种持续的改进努力将进一步加强我们的算法在不同场景下的可靠性和适应性。

参考文献

[1] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2019.

[2] Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 1998.

[3] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530, 2006.

[4] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[6] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2011.

[7] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International conference on artificial intelligence and statistics*, pages 803–812. PMLR, 2018.

[8] Paul Erdős and Robin J Wilson. On the chromatic index of almost all graphs. *Journal of combinatorial theory, series B*, 23(2-3):255–257, 1977.

[9] Miroslav Fiedler. Eigenvalues of nonnegative symmetric matrices. *Linear Algebra and its Applications*, 9:119–142, 1974.

[10] Suyog Gupta, Wei Zhang, and Fei Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 171–180. IEEE, 2016.

[11] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, and Kurt Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2592–2600, 2016.

[12] Dusan Jakovetic, Dragana Bajovic, Anit Kumar Sahu, and Soummya Kar. Convergence rates for distributed stochastic optimization over random networks. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4238–4245. IEEE, 2018.

[13] Zhanhong Jiang, Aditya Balu, Chinmay Hegde, and Soumik Sarkar. Collaborative deep learning in fixed topology networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[14] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[15] Soummya Kar, Saeed Aldosari, and José MF Moura. Topology for distributed inference on graphs. *IEEE Transactions on Signal Processing*, 56(6):2609–2613, 2008.

[16] Soummya Kar and José MF Moura. Sensor networks with random links: Topology design for distributed consensus. *IEEE Transactions on Signal Processing*, 56(7):3315–3326, 2008.

[17] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*, pages 3478–3487. PMLR, 2019.

[18] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.

[19] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems*, 30, 2017.

[20] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[21] Ilan Lobel and Asuman Ozdaglar. Distributed subgradient methods for convex optimization over random networks. *IEEE Transactions on Automatic Control*, 56(6):1291–1306, 2010.

[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[23] Jayadev Misra and David Gries. A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.

[24] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[25] Kevin Scaman, Francis Bach, Sébastien Bubeck, Laurent Massoulié, and Yin Tat Lee. Optimal algorithms for non-smooth distributed optimization in networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[26] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[27] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966, 2015.

[28] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[29] Hanlin Tang, Shaoduo Gan, Ce Zhang, Tong Zhang, and Ji Liu. Communication compression for decentralized training. *Advances in Neural Information Processing Systems*, 31, 2018.

[30] Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. d^2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pages 4848–4856. PMLR, 2018.

[31] Zaid J Towfic, Jianshu Chen, and Ali H Sayed. Excess-risk of distributed stochastic learners. *IEEE Transactions on Information Theory*, 62(10):5753–5785, 2016.

[32] Konstantinos Tsianos, Sean Lawlor, and Michael Rabbat. Communication/computation tradeoffs in consensus-based distributed optimization. *Advances in neural information processing systems*, 25, 2012.

[33] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[34] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. *Proceedings of Machine Learning and Systems*, 1:212–229, 2019.

[35] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *The Journal of Machine Learning Research*, 22(1):9709–9758, 2021.

[36] Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[37] Ran Xin, Usman A Khan, and Soummya Kar. Variance-reduced decentralized stochastic optimization with accelerated convergence. *IEEE Transactions on Signal Processing*, 68:6255–6271, 2020.

[38] Kun Yuan, Qing Ling, and WotaOn nonconvex decentralized gradient descento Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26(3):1835–1854, 2016.

[39] Jinshan Zeng and Wotao Yin. On nonconvex decentralized gradient descent. *IEEE Transactions on signal processing*, 66(11):2834–2848, 2018.