软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007706] [CSTR: 32375.14.jos.007706]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于大规模并行模型的完全动态单源最短路径算法*

王迟蕾[1,2,3,4], 华强胜[1,2,3,4], 金 海[1,2,3,4]

[1](大数据技术与系统国家地方联合工程研究中心 (华中科技大学), 湖北 武汉 430074)

[2](服务计算技术与系统教育部重点实验室 (华中科技大学), 湖北 武汉 430074)

[3](集群与网格计算湖北省重点实验室 (华中科技大学), 湖北 武汉 430074)

[4](华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)

通信作者: 华强胜, E-mail: qshua@hust.edu.cn

摘　要: 研究大规模并行计算模型中动态单源最短路径问题. 在该模型中, 每台机器内存大小为 $O(n^{\alpha})$, n 表示图中顶点数, $\alpha \in (0,1)$ 是一个常数. 主要考虑带整数边权重的有向或无向图, 其中边权重最大值与最小值之比的上限为 $\lceil poly(\log n) \rceil$. 图的更新操作包括单边插入、删除以及权重改变. 针对上述问题, 结合多项式矩阵逆方法、路径分解策略以及并行线性代数算法; 同时, 通过设计一种高效的并行多项式乘法求解算法提出了一个随机并行动态单源最短路径算法. 该算法的更新轮数复杂度为 $O(\alpha^{-1}\log n)$, 总内存需求为 $\tilde{O}(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})$. 其中, 更新轮数复杂度指每次图中边发生变化后, 为重新计算从源点到其他受影响顶点之间的距离, 算法所需同步的迭代轮数; $\tilde{O}(\cdot)$ 隐去了对数多项式因子, ω 为快速矩阵乘法的算术复杂度 ($O(n^{\alpha})$) 指数. 相较于现有方法至少需要 $poly(\log n)$ 轮数复杂度的并行静态单源最短路径算法, 所提并行动态精确单源最短路径算法显著降低了轮数复杂度.

关键词: 动态图; 多项式矩阵; 矩阵乘法; 大规模并行计算模型

中图法分类号: TP303

中文引用格式: 王迟蕾, 华强胜, 金海. 基于大规模并行模型的完全动态单源最短路径算法. 软件学报. http://www.jos.org.cn/1000-9825/7706.htm

英文引用格式: Wang CL, Hua QS, Jin H. Fully Dynamic Single-source Shortest Paths Algorithm Based on Massively Parallel Model. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7706.htm

Fully Dynamic Single-source Shortest Paths Algorithm Based on Massively Parallel Model

WANG Chi-Lei[1,2,3,4], HUA Qiang-Sheng[1,2,3,4], JIN Hai[1,2,3,4]

[1](National Engineering Research Center for Big Data Technology and System (Huazhong University of Science and Technology), Wuhan 430074, China)

[2](Key Laboratory of Services Computing Technology and System (Huazhong University of Science and Technology), Ministry of Education, Wuhan 430074, China)

[3](Cluster and Grid Computing Lab (Huazhong University of Science and Technology), Wuhan 430074, China)

[4](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: The dynamic single-source shortest paths problem is studied in the massively parallel computation (MPC) model, where the memory size of each machine is $O(n^{\alpha})$. Here, n is the number of vertices in the graph, and $\alpha \in (0,1)$ is a constant. This study primarily considers directed or undirected graphs with integer edge weights, where the ratio between the maximum and minimum edge weights is upper-bounded by $\lceil poly(\log n) \rceil$. The graph update operations include single-edge insertion, deletion, and weight change. To address this problem, this study combines the polynomial matrix inverse method, the path decomposition strategy, and parallel linear algebra algorithms. In addition, by designing an efficient parallel polynomial multiplication algorithm, a randomized parallel dynamic single-source

* 基金项目: 国家自然科学基金 (62372202, 62461160333)
收稿时间: 2024-08-08; 修改时间: 2026-04-03; 采用时间: 2026-05-21; jos 在线出版时间: 2026-08-12

shortest paths algorithm is proposed. The algorithm achieves an update round complexity of $O(\alpha^{-1}\log n)$ and a total memory requirement of $\tilde{O}(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})$. Here, the update round complexity refers to the number of synchronous iteration rounds required by the algorithm to recompute, after each edge change in the graph, the distances from the source to other affected vertices. $\tilde{O}(\cdot)$ suppresses polylogarithmic factors, and ω represents the exponent of the arithmetic complexity of fast matrix multiplication on matrices of size $O(n^{\alpha})$. Compared with existing parallel static single-source shortest paths algorithms that require at least $poly(\log n)$ round complexity, the proposed algorithm significantly reduces the round complexity.

Key words: dynamic graph; polynomial matrix; matrix multiplication; massively parallel computation (MPC) model

近年来, 大规模并行计算模型 (massively parallel computation model, MPC 模型)[1−3]在图算法研究中受到广泛关注, 譬如, 用于求解最小生成树 (minimum spanning tree, MST) [4]、图连通性 [1]、最小割 [5]以及近似最短路径 [6,7]等. 然而, 现有 MPC 算法大多基于静态图, 每当图中边或顶点发生变化时, 都需重新计算整个图, 从而造成资源和时间浪费.

与静态图算法相比, 动态图算法[8–12]只需处理规模较小的实时变化数据. 动态图算法利用高效数据结构, 快速处理一系列图更新 (边或顶点的插入、删除以及边权重的修改) 和查询操作, 从而维护图的特定性质. 在最短路径问题中, 动态最短路径算法用于维护顶点间距离或最短路径. 迄今为止, 单机上已有大量动态图算法研究 [13–16], 这些算法的时间复杂度普遍低于对应的静态图算法. 然而, 在大数据时代, 单机计算已经无法满足海量数据处理需求. 那么, 单机动态图算法的优势能否扩展至 MPC 模型? 本文对此给出肯定回答. 已有研究表明, MPC 模型中的动态图算法[1–4]的轮数复杂度优于对应的并行静态图算法. 例如, Italiano 等人[3]设计了一种用于求解动态最大匹配的 MPC 算法, 在单条边删除或插入时, 仅需 $O(1)$ 更新轮数复杂度, 而对应的静态 MPC 算法重新计算图的最大匹配则需 $O(\log^{1/2} n)$ 轮, 其中 n 表示给定图的顶点数. 因此, 研究 MPC 模型中的动态图算法具有重要意义和必要性.

单源最短路径 (single-source shortest path, SSSP) 算法作为一个基础图算法, 已有工作针对不同并行计算模型展开研究, 包括 PRAM 模型 [17]、Congest 模型 [18,19]和 MPC 模型 [6], 这凸显了该问题的重要性. 尽管过去 40 年针对动态最短路径问题的研究已取得丰硕成果 [9–11,20,21], 但据我们所知, 目前尚无 MPC 模型下的动态单源最短路径的相关研究.

因此, 本文研究 MPC 模型下加权稠密图中的完全动态精确 SSSP 问题. 其中, 完全动态指更新操作为单条边插入、删除以及单条边权重改变. 本文主要目标为设计一个具有低轮数复杂度的 MPC 算法, 能够有效更新源点到其他顶点之间距离. 总体而言, 所提并行完全动态 SSSP 算法借鉴了文献 [12] 设计的顺序完全动态近似 SSSP 算法的基本思想, 即将距离的计算与更新分为两部分: 短边最短路径长度 (由较少边组成的路径) 和长边最短路径长度 (由较多边组成的路径). 主要挑战如下.

- 文献 [12] 在计算和更新短边距离时, 采用了多项式矩阵求逆方法, 并在其中利用快速傅里叶变换 (fast Fourier transform, FFT) 计算两个多项式乘法. 而现有并行 FFT 算法的轮数复杂度为 $O(\log n)$, 难以满足 MPC 模型中的动态 SSSP 更新效率.
- 文献 [12] 在计算和更新长边距离时, 需在基于短边距离构造的新图上运行 Dijkstra 算法. 而在 MPC 模型中直接运用 Dijkstra 算法会导致很高的轮数复杂度.

本文同样利用多项式矩阵求逆来更新顶点之间短边最短路径长度, 但在更新短边最短路径长度时, 设计了新的并行多项式乘法来降低更新轮数复杂度. 更新长边最短路径长度时, 利用半环上向量-矩阵乘法和路径平方策略等方法计算源顶点到其他顶点的距离. 同时, 还为半环上向量-矩阵乘法等子程序设计新的高效 MPC 算法, 从而降低 MPC 模型中动态精确 SSSP 算法的更新轮数复杂度和总内存需求.

此外, 我们未采用文献 [12] 在设计顺序动态近似 SSSP 算法时, 为分析最坏情况下的更新时间而普遍使用的标准技术, 即每隔固定次数的边更新后重新处理整个图. 由于考虑 MPC 模型, 因此无需在更新次数达到某个固定值后重新处理图. 我们将预处理算法与单边更新算法分开, 这意味着原始图仅需一次预处理, 此后每次边更新只需执行单边更新算法. 其中预处理算法类似于静态图算法, 目的是构建高效数据结构; 在图中边发生变化时, 更新算法利用这些高效数据结构来更新源点到其他顶点之间的距离.

本文主要贡献如下.

● 提出 MPC 模型的完全动态精确 SSSP 算法 (详见第 4.3 节和算法 6), 用于更新源点到其他顶点的距离, 并将该算法推广至完全动态精确所有节点对最短路径 (all pairs shortest paths, APSP) 问题的距离更新.

● 与现有工作相比, 所提并行动态精确 SSSP 算法具有更低的更新轮数复杂度 (详见表 1).

● 针对 MPC 模型中两个多项式乘积问题, 设计了常数轮的并行算法, 其轮数复杂度远低于并行 FFT 算法 (详见第 4.1 节).

表 1　静态并行 SSSP 算法与所提并行动态 SSSP 算法的复杂度比较

算法	问题类型	更新轮数	查询轮数	总内存	图类型	算法类型
文献[6]	$(1+\varepsilon)$-SSSP	$O(2^{\tilde{O}(\sqrt{\log n})}/\alpha)^{*}$	$O(1)$	$O(n^2)$	无向	随机
文献[22]	$(1+\varepsilon)$-SSSP	$(\log n)^{O(1/\rho)}$**	$O(1)$	$\tilde{O}(n^{2+\rho})$	无向	确定
文献[23]	精确SSSP	$O(n^{1/2+o(1)})$	$O(1)$	$O(n^{3/2})$	有向	随机
DPV[12]	动态 $(1+\varepsilon)$-SSSP	$O(n^{1.823}/\varepsilon^2)$	$O(\varepsilon^{-1}n^{0.45})$	$O(n^{\alpha})^{\dagger}$	有向/无向	随机
DPV[24]	动态精确SSSP	$O(n^{1.969})$	$O(n^{1.969})$	$O(n^{\alpha})^{\dagger}$	有向	确定
本文	动态精确SSSP	$O(\alpha^{-1}\log n)$	$O(\alpha^{-1})$	$\tilde{O}(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})^{\ddagger}$	有向/无向	随机

注: (1) DPV: 直接并行化版本; (2) *: $\alpha\in(0,1)$ 为内存参数; (3) **: $\rho\in(0,1/2)$; (4) †: $O(1)$ 台机器的总内存; (5) ‡: $\omega\simeq 2.3728$ 为快速矩阵乘法的算术复杂度指数, 总内存随 $\alpha\in(0,1)$ 从 $\tilde{O}(n^{2.276})$ 到 $\tilde{O}(n^{2.5296})$ 变化

接下来, 将所设计的并行完全动态 SSSP 算法与现有 MPC 模型中静态 SSSP 算法进行比较, 如表 1 所示. 表 1 同时给出了文献 [12] 的顺序动态 $(1+\varepsilon)$-近似 SSSP 算法经直接并行化后所需的更新轮数复杂度和总内存需求. 由于预处理算法在原始图上仅使用一次, 表 1 只呈现并行动态 SSSP 算法的更新轮数复杂度和查询轮数复杂度. 其中, 预处理轮数复杂度表示对原图进行预处理以构建高效数据结构所需轮数, 而查询轮数复杂度定义为查询源点到其他任意顶点所需轮数.

如表 1 所示, 在单边更新下, 所提并行完全动态 SSSP 算法所需更新轮数复杂度 (表 1 最后一行第 3 列) 显著低于现有并行静态 $(1+\varepsilon)$-近似 SSSP 算法 (表 1 第 1、2 行) 或精确 SSSP 算法 (表 1 第 3 行), 而查询轮数复杂度差别很小, 均为常数 (表 1 第 1–3 行第 4 列). 本文结果优于 MPC 模型中的现有静态 SSSP 算法. 此外, 所提并行动态 SSSP 算法所需总内存 (不超过 $\tilde{O}(n^{2.5296})$) 与 Elkin 等人[22]的并行静态 $(1+\varepsilon)$-近似 SSSP 算法 (表 1 第 2 行第 5 列) 所需总内存 (不超过 $\tilde{O}(n^{2.5})$) 差距很小. 然而, 相较于 Cao 等人[23]提出的并行精确 SSSP 算法所需总内存 ($O(n^{3/2})$), 所提并行动态 SSSP 算法内存需求增大了 $O(n^{1.0296})$ 倍. 这是因为 Cao 等人[23]将精确 SSSP 问题归约为近似 SSSP 问题, 而后者通常通过构造边数较少的新图 (详见第 3.1 节) 来降低存储内存需求.

此外, 表 1 还给出了文献 [12] 的顺序动态 $(1+\varepsilon)$-近似 SSSP 算法以及文献 [24] 的顺序动态精确 SSSP 算法的直接并行化结果. 这两个结果均利用了 Italiano 等人[3]的证明: 在每轮使用 $O(1)$ 台机器计算的情况下, 一个顺序动态图算法的预处理时间、更新时间和查询时间分别等于对应 MPC 算法所需的预处理轮数、更新轮数和查询轮数. 尽管这些直接并行化的算法每轮内存需求 ($O(n^{\alpha})$) 很小, 但这两个并行动态算法的更新轮数复杂度很高, 导致在 MPC 模型中效率不高. 需要指出, 文献 [12] 的动态近似 SSSP 算法需 $O(\varepsilon^{-1}n^{0.45})$ 查询轮数复杂度 (表 1 第 4 行第 4 列), 文献 [24] 的动态精确 SSSP 算法需 $O(n^{1.969})$ 查询轮数复杂度 (表 1 第 5 行第 4 列), 而所提并行动态 SSSP 算法仅需常数轮便可查询源点到任意一个顶点的距离. 这是因为上述两个顺序动态算法都采用了 Dijkstra 算法.

所提并行动态 SSSP 算法的主要缺点是仅能更新距离, 无法更新最短路径. 此外, 虽然本文与 Cao 等人[23]均考虑整数边权重, 但所提并行动态 SSSP 算法适用边权重范围更窄. 具体而言, 最大边权重与最小边权重的比值为 $\lceil poly(\log n)\rceil$, 这表明所提并行完全动态 SSSP 算法不适用于边权重比值很大的动态图.

本文第 1 节介绍动态单源最短路径的研究现状. 第 2 节阐述基础知识, 包括模型介绍、问题定义以及多项式矩阵等. 第 3 节介绍在 MPC 模型中更新动态 SSSP 问题所需关键技术和算法子程序. 第 4 节设计 MPC 模型中多项式乘法的高效并行算法, 求解动态 SSSP 问题所设计的预处理算法、单边更新算法以及主要结果. 第 5 节对第

4 节中设计的并行完全动态 SSSP 算法进行修改, 以用于更新动态 APSP 问题的距离. 第 6 节总结全文.

1 相关工作

1.1 动态单源最短路径研究现状

目前, 大多数研究致力于降低单机环境下动态近似 SSSP 问题的总更新时间复杂度 (即执行多个更新操作所需总时间). Even 等人[20]提出的 ES (Even-Shiloach) 树已成为构建动态近似最短路径算法的基本组件. ES 树表明, 动态精确 SSSP 可以在 $O(mn)$ 总时间内完成更新, 其中 m 表示边的数量. 与 Dijkstra 算法 (运行时间为 $O(m+n\log n)$) 相比, 当更新操作数量为 m 时, ES 树更具竞争优势.

近似算法通常比精确算法具有更低的时间复杂度. Roditty 等人[21]证明, 对于递减 (更新操作为边删除或边权重增大) 精确 SSSP 问题, 实现低于 $O(mn)$ 时间复杂度极具挑战性. 因此, 研究人员倾向于寻找动态 $(1+\varepsilon)$-近似 SSSP 的高效解决方案. 早期研究主要结合 ES 树和 Thorup-Zwick 距离预言机 (distance oracles)[13,25]: ES 树用于计算短边最短路径, 而 Thorup-Zwick 距离预言机用于计算长边最短路径. 后来, Bernstein 等人[26]提出了第 1 个确定性算法, 采用一种新颖且更简单的方法, 针对不同度数的顶点构建相应稀疏图. 通过在每个稀疏图上运行 ES 树, 该算法只需 $\tilde{O}(n^2)$ 总时间即可更新无向无权图中 $(1+\varepsilon)$-近似 SSSP. 随后, 将这种方法推广至无向加权图[27]和稀疏图[28]. 其他研究包括将 Bernstein 等人[28]的方法和稀疏图的聚类技术以及近似 SSSP 的最短路径[29,30]相结合. 本文不讨论那些使用了类似于 Bernstein 等人[27]方法的增量 (更新操作为边的插入或者边权重的降低) 动态 SSSP 算法[31]以及有向图中动态近似 SSSP[11]的研究. 因为它们关注 SSSP 算法及其他方面的改进, 与本文研究关系不大.

将上述基于总更新时间复杂度的算法并行化面临巨大挑战, 因为 MPC 算法更侧重于优化算法深度而非算术复杂度. 这意味着, 即使总更新时间显著降低, 算法深度并不一定随之降低. 单机中动态图算法的总更新时间复杂度优势可能无法扩展至 MPC 模型中, 因为 MPC 模型的主要目标是降低轮数复杂度. 因此, 我们需要寻找具有最坏情况更新时间保证的动态 SSSP 算法设计方法.

近年来, Bernstein 等人[15]设计了一种算法, 以 $O(1)^k\log^3 n$ 的最坏情况更新时间维护具有 $\tilde{O}(n^{1+1/k})$ 条边的 $(2k-1)$-跨度图, 该算法近似比较大且仅适用于无向图. 与此同时, 文献 [12] 提出一种随机算法, 采用多种代数方法解决动态 $(1+\varepsilon)$-近似最短路径问题. 其核心技术可追溯到 Sankowski[14]用于求解动态传递闭包问题的动态矩阵求逆方法. 随后, Sankowski[9]将动态矩阵求逆扩展到具有多项式元素的矩阵. 其核心思想是利用多项式矩阵求逆来计算短边最短路径长度, 并结合路径分解方法[32]和半环上矩阵乘法来计算长边最短路径长度. 基于此, Sankowski 设计了一个随机算法, 在具有整数边权重的无向图中, 实现 $O(n^{1.982})$ 动态距离更新时间和 $O(n^{1.288})$ 查询时间. 受 Sankowski[9]工作启发, 文献 [12] 在计算多项式矩阵求逆后, 利用路径分解方法构建一个边数较少的新图, 并在该图上执行 Dijkstra 算法. 最终, 该动态 $(1+\varepsilon)$-近似 SSSP 算法在单边更新下的最坏情况更新时间为 $\tilde{O}(n^{1.823}/\varepsilon^2)$. 在稠密图 (即 $m=O(n^2)$) 场景下, 该方法比直接在原图上运行 Dijkstra 算法更为高效. 最近, 文献 [24] 借鉴了文献 [12] 求解动态 $(1+\varepsilon)$-近似 SSSP 问题的思路, 设计了首个完全动态精确 SSSP 算法, 其最坏更新时间和查询时间均为 $O(n^{1.969})$. 不同之处在于, 文献 [24] 提出一个确定性算法, 即在求解和更新短边距离时, 将顶点集划分为不同子集以得到不同子图, 并在这些子图上运用多项式矩阵逆; 而在求解长边距离时, 则在基于短边距离构造的新图上运行 Dijkstra 算法, 从而得到源点到所有其他顶点的距离.

1.2 基于 MPC 模型的并行最短路径算法研究现状

迄今为止, 关于 MPC 和 PRAM 模型中静态 SSSP 算法的研究主要集中于求解近似解. Dinitz 等人[6]提出了一种求解静态 $(1+\varepsilon)$-近似 SSSP 问题的 MPC 算法, 其轮数复杂度高达 $O(1/\alpha)\cdot 2^{\tilde{O}(\sqrt{\log n})}$, 其中 $\varepsilon\in(0,1)$ 是一个常数, α 是介于 0–1 之间的常数, 表示 MPC 模型中每台机器内存为 $O(n^{\alpha})$. 尽管 Andoni 等人[17]提出了轮数复杂度低至 $O(poly(\log n))$ 的 MPC 算法, 但该算法仅适用于 $(1+\varepsilon)-s-t$ 最短路径问题. 然而, 上述所有并行最短路径算法均为随机算法. 至于确定性算法, 目前仅在 PRAM 模型中研究了 $(1+\varepsilon)$-近似 SSSP 算法[22,33]. Goodrich 等人[34]证明了 MPC 模型和 PRAM 模型之间的关系, 表明 MPC 模型中算法的轮数复杂度通常不低于 PRAM 模型中相同算法的

深度, 在典型参数下两者仅相差一个常数因子. 因此, MPC 模型中确定性 $(1+\varepsilon)$-近似 SSSP 算法需 $O(poly(\log n))$ 轮. 最近, Cao 等人[23]提出了一种针对有向图精确 SSSP 问题的随机 PRAM 算法, 其运行深度为 $O(n^{1/2+o(1)})$. 该研究通过将精确 SSSP 问题归约为近似 SSSP 问题 (即在构建的新图上计算距离估计) 来求解. 其图构建方法与 Dinitz 等人[6]的研究类似, 详见第 3.1 节.

此外, 虽有少量关于 GPU[35]和共享内存 CPU[36]上动态 SSSP 算法的应用研究, 但这些研究主要关注动态 SSSP 算法的平均运行时间. 鉴于本文侧重于理论模型及算法的最坏情况时间复杂度, 故不再对此展开详述.

2 预备知识

本节阐述 MPC 模型, 问题定义以及多项式矩阵等预备知识.

2.1 模型介绍与问题定义

● MPC 模型: 大规模并行计算模型 (MPC 模型) 最初由 MapReduce 模型发展而来[37]. 在 MPC 模型中, 算法在同步轮中执行. 具体而言, 每轮开始时, 所有机器通过相互通信获取所需数据. 随后, 所有机器进行本地独立计算且彼此不通信. 每轮结束时, 所有机器交换信息, 以准备下一轮计算. 每台机器从其他机器发送或接收的数据总量不超过单台机器内存大小 S.

MPC 模型包含 3 种类型的内存设置: 强超线性内存、近线性内存以及强亚线性内存. 其中, 在强亚线性内存的 MPC 模型中设计算法最具挑战性. 本文考虑满足 $S=O(N^{\alpha})$ 的强亚线性内存, 其中 S 表示单台机器的内存大小, N 表示输入规模 (通常指图的边数或顶点数), 参数 $\alpha\in(0,1)$. 我们设定输入规模 N 为图的顶点数量 n. 此外, 模型假设机器间的通信网络为全互联结构.

● 复杂度衡量: 在 MPC 模型中, 主要目标是降低所设计并行动态算法的轮数复杂度. 次要目标是尽可能地减小运行动态图算法所需总内存.

定义 1. 问题定义. 考虑一个加权无向或有向稠密图 $G=(V,E)$, 其中边权重为正整数, 且最大边权与最小边权之比为 $\lceil poly(\log n)\rceil$. 设 $n=|V|$ 表示顶点数, $m=|E|$ 表示边数, $s\in V$ 表示图 G 的源点. 在单边更新 (包括单边插入或删除以及单边权重修改) 操作下, 动态精确 SSSP 问题旨在维护源点 s 到图 G 中其他顶点的距离.

我们的目标是设计一种低轮数复杂度 MPC 算法以高效求解精确动态 SSSP 问题.

2.2 多项式矩阵

● 矩阵和矩阵乘法: 定义 $\mathbb{I}$ 为单位矩阵, A 为给定图 $G=(V,E)$ 的邻接矩阵. 设 $I\subset[n]:=\{1,2,\ldots,n\}$, $J\subset[n]:=\{1,2,\ldots,n\}$. $A_{I,J}$ 是矩阵 A 的子矩阵, 其行索引在 I 中, 列索引在 J 中. 当 $I\subset[n]$ 且 $J=\{j\}$, $j\in[n]$(或 $J\subset[n]$ 且 $I=\{i\}$, $i\in[n]$) 时, $A_{I,J}$ 表示 A 的第 j 列 (或第 i 行). 对于矩阵乘法, 定义 $\omega(a,b,c)$ 为 $n^{\omega(a,b,c)}$ 的指数, 其中 $n^{\omega(a,b,c)}$ 表示计算 $n^a\times n^b$ 矩阵与 $n^b\times n^c$ 矩阵的乘积的计算复杂度. 当 $a=b=c=1$ 时, 使用快速矩阵乘法 (如 Strassen 矩阵乘法) 可使 $\omega(a,b,c)<3$. 目前已知矩阵乘法的最小指数 $\omega\approx 2.3728$ [38].

● 模 X^d 的多项式[12]: 符号 $\mathbb{F}[X]$ 表示带有变量 X 的某个域. $p,q\in\mathbb{F}[X]$ 为两个多项式, 且 $degree(p)$, $degree(q)\leqslant a$. 两个次数不超过 a 的多项式相加或相减的计算复杂度为 $O(a)$. 对于任意两个多项式的乘法, 现有最优算法为快速傅里叶变换 (FFT), 其时间复杂度为 $O(a\log a)$. 本文设计的算法在 $\mathbb{F}[X]/\langle X^d\rangle$ 域上执行, 而非 $\mathbb{F}[X]$. 这意味着, 若多项式 $p=c-bX$, 其中 $c\in\mathbb{F}\backslash\{0\}$ 且 $b\in\mathbb{F}[X]/\langle X^d\rangle$, 则 p 可逆, 且有:

$$p^{-1}=c^{-1}\sum_{k=0}^{d-1}(bX/c)^k=c^{-1}\prod_{k=0}^{\log n}(1+(bX/c)^{2^k}).$$

● 多项式矩阵: 考虑多项式矩阵 $M=\sum_{k=0}^{d-1}M^{[k]}X^k\in(\mathbb{F}[X]/\langle X^d\rangle)^{n\times n}$ 和多项式向量 $v=\sum_{k=0}^{d-1}v^{[k]}X^k\in(\mathbb{F}[X]/\langle X^d\rangle)^n$. 矩阵 M 或向量 v 的每个元素均为 $\mathbb{F}[X]/\langle X^d\rangle$ 中多项式. 设 $M^{[k]}$ 是 X^k 的系数矩阵, $v^{[k]}$ 是 X^k 的系数向量. 值得注意的是, $M^{[k]}\in(\mathbb{F}[X]/\langle X^d\rangle)^{n\times n}$ 且 $v^{[k]}\in(\mathbb{F}[X]/\langle X^d\rangle)^n$. 若两个 $n\times n$ 矩阵相乘的计算复杂度为 $n^{\omega(1,1,1)}$, 则两个同阶多项式矩阵相乘的计算复杂度为 $O(dn^{\omega(1,1,1)})$. 然而, 由于计算复杂度并非 MPC 模型的主要复杂度衡量指标, 两个多项式矩阵相

乘所需轮数需同时考虑矩阵相乘和多项式元素相乘的轮数开销, 相关细节详见第 4.2 节和第 4.3 节.

当多项式矩阵 $M=\mathbb{I}-B$, 其中 $B\in(\mathbb{F}[X]/\langle X^d\rangle)^{n\times n}$ 时, M 可逆且 $M^{-1}=\sum_{k=0}^{d-1}B^k=\prod_{k=0}^{\log n}(1+B^{2^k})$, 这可由上述多项式求逆公式推导得出.

3 主要技术与方法

本节介绍 MPC 模型中用于更新动态 SSSP 的关键引理与技术. 首先, 第 3.1 节通过一个简单例子说明所提方法与现有并行近似 SSSP 算法的差异, 以阐明所提算法的设计思路. 然后, 第 3.2 节和第 3.3 节分别给出所提并行动态 SSSP 算法所需引理和子程序.

3.1 一个简单例子

本文研究 MPC 模型中动态精确 SSSP 问题, 主要采用 Sankowski[9]和文献 [12] 提出的多项式矩阵求逆方法. 为便于理解, 本节首先将所提并行动态 SSSP 算法与 MPC 模型中静态近似 SSSP 算法进行对比.

图 1 展示了 Dinitz 等人[6]在 MPC 模型中求解静态 SSSP 问题的过程. 首先, 构造一组边, 使任意两个顶点间的最短路径包含至多 h 条边. 这些边权重通过超聚类和互连方法[6]确定为某些顶点对之间的距离, 并将其添加至原始图中. 随后, 在扩展图上运行 Bellman-Ford 算法, 得到从源点 s 到其他顶点的距离. 由于 s 到这些顶点的路径边数上限为 h, 该过程需 h 轮完成. 定义 d^h 为路径包含至多 h 条边的两个顶点间距离. 则扩展图中两个顶点的距离 d^h 与原始图中对应顶点的距离 d 满足 $d\leqslant d^h\leqslant(1+\varepsilon)\cdot d$.

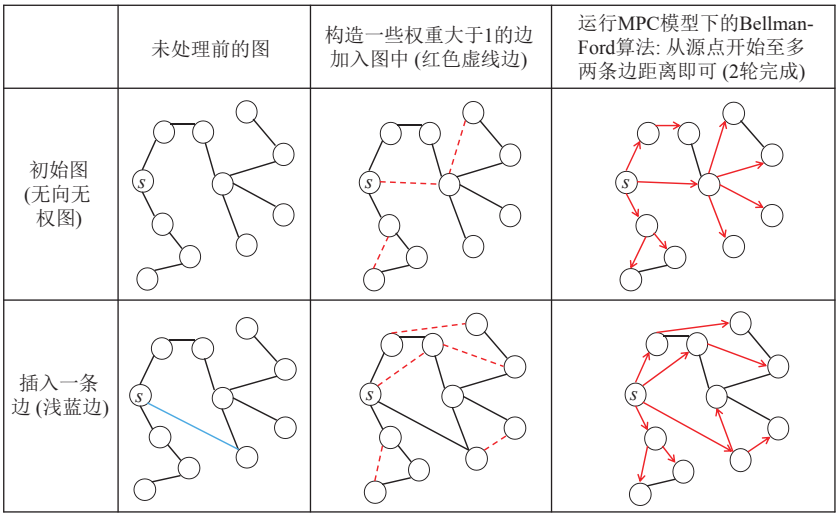

图 1 Dinitz 等人[6]在 MPC 模型下计算无向无权图中静态近似 SSSP 的主要步骤

在图 1 中, 红色虚线表示新构造的边. 从 s 到其他顶点的近似距离仅需两轮即可算出. 当插入一条边 (浅蓝色边) 时, 算法需删除所有已构造的边, 重新构造符合条件的边, 并重新计算距离. 这一过程可能非常耗时.

与 Dinitz 等人[6]的方法不同, 我们利用多项式矩阵求逆作为核心组件来更新顶点间的短边最短路径长度. 如图 2 所示, 首先将原始图转化为多项式矩阵 (图 2 第 2 行第 3 列). 随后, 计算该多项式矩阵的逆, 其对应于顶点间不超过 5 条边的最短路径长度 (图 2 第 2 行第 4 列). 从源点 s 到其他顶点的多项式向量由红色虚线框中的元素组成 (图 2 ④). 通过寻找变量 X 具有非零系数的最小次数, 即可获得从 s 到某个顶点的至多 5 条边的最短路径长度 (图 2 ③). 在图 2 中, 从 s 到任何其他顶点的路径边数不超过 5. 因此, 通过多项式矩阵求逆得到的距离即为从 s 到其他顶点的距离.

当插入一条浅蓝色边时, 只需将该插入边的信息转换为一个矩阵, 该矩阵仅含一个非零多项式元素 (图 2 ⑤).

利用 Sherman-Morrison 恒等式 (详见第 3.2 节引理 2), 可在一轮内得到一个多项式矩阵 (图 2 ⑥). 然后, 从原始多项式矩阵的逆中减去该矩阵 (图 2 第 4 行第 3 列中带有橙色元素的矩阵). 最后, 得到从 s 到其他顶点的多项式向量 (图 2 ⑦).

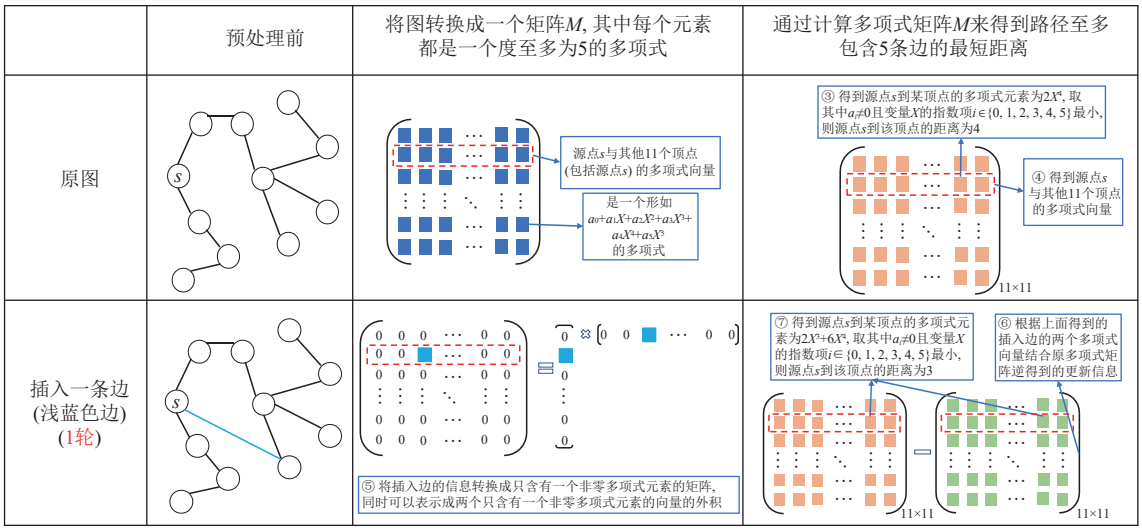

图 2　本文所设计 MPC 模型中完全动态 SSSP 算法的重要组成部分——多项式矩阵逆

总之, 不同于 Dinitz 等人[6]在图 1 中需重新构造边并重新计算图, 本文方法仅需关注插入边对原多项式矩阵的影响. 这种方法不仅可以降低计算距离的时间, 更有效避免了冗余计算.

3.2　必要的引理

第 1 个策略是将距离问题归约为多项式矩阵求逆, 该方法最早由 Sankowski[9,14]提出, 用于求解无向图的动态传递闭包和动态最短路径长度问题. 基于 Sankowski 的工作, 文献 [12,39] 分别针对动态近似最短路径问题和敏感距离预言机 (sensitive distance oracle) 问题, 提出了两个相似的引理. 本文研究主要基于文献 [12] 的方法, 具体归约策略如下.

引理 1. 归约策略[12]. 给定一个正整数 h 和一个有向图 $G=(V,E)$, 其中 $(u,v)\in E$, 边权重 $w_{u,v}$ 为整数且 $w_{u,v}\in[[poly(\log n)]]$. 设 $B(G)\in(\mathbb{F}[X]/\langle X^d\rangle)^{n\times n}$ 满足如下条件: 对于边 $(u,v)\in E$, 有 $B(G)_{u,v}=a_{u,v}X^{w(u,v)}$; 对于每个顶点 $u\in V$, 有 $B(G)_{u,u}=a_{u,u}X$; 其他情况, $B(G)_{u,v}=0$. 令 $a_{u,v}$ 和 $a_{u,u}$ 是从 $\mathbb{F}$ 中独立且均匀抽样得到的元素. 那么, 矩阵 $M=\mathbb{I}-B(G)$ 可逆, 且对于任意 $u,v\in V$ 和 $h\in[0,d)$, 满足以下性质的概率至少为 $1-hn^2/\mathbb{F}$:

$$(M^{-1})^{[h]}_{u,v}\neq 0\Leftrightarrow dist(u,v)\leqslant h.$$

该引理阐明距离和多项式矩阵求逆之间的关系. 具体而言, 多项式矩阵逆 M^{-1} 展示了任意两个顶点之间所有可能路径的总和, 其距离为 h. 变量 X 的指数项表示任意顶点对之间距离. 读者可在文献 [12] 中获取更详细的信息以及该引理的证明.

接下来, 介绍 Sherman-Morrison 恒等式. 文献 [12] 利用该恒等式处理每次边更新, 这也是求解 MPC 模型中单边更新下动态精确 SSSP 问题的第 2 个策略.

引理 2. Sherman-Morrison 恒等式[12]. 给定一个 $n\times n$ 矩阵 M 和一个 $1\times n$ 向量 u,v, 则有:

$$(M+u^{\mathrm{T}}v)^{-1}=M^{-1}-M^{-1}u^{\mathrm{T}}(1+vM^{-1}u^{\mathrm{T}})^{-1}vM^{-1}.$$

Sherman-Morrison 恒等式说明, 若对可逆矩阵 M 施加一个微小扰动, 例如添加一个外积 $u^{\mathrm{T}}v$, 则所得矩阵 $M+u^{\mathrm{T}}v$ 仍然可逆, 并满足上述关系式. 该性质在 MPC 模型中单边更新条件下计算动态 SSSP 至关重要.

第 3 个引理给出用于路径分解的随机抽样策略, 常用于低算术复杂度的顺序动态最短路径算法设计.

引理 3. 随机抽样技术[33]. 给定一个图 $G=(V,E)$, 其中 $n=|V|$, $H\subset V$ 为从集合 V 中均匀随机抽样得到的顶点集. 如果顶点 v 和顶点 u 之间存在一条至多包含 $cn\log n/|H|$ 个顶点的路径, 则集合 H 中至少有一个顶点属于该路径的概率为 $1-n^{-c}$. 参数 $c>0$ 为常数.

3.3 算法子程序

首先, 介绍几种现有 MPC 算法, 它们将在第 4 节的并行完全动态 SSSP 算法中发挥作用.

● *Find-Minimum*$(S,P(a_1),P(a_n))$[6]: 在 MPC 模型中, 查找由 n 个可比较值构成的集合 S 中的最小值. 集合 S 中的元素顺序存储在一组连续机器上, 满足第 1 台机器 $P(a_1)$ 存储第 1 个元素 a_1, 最后一台机器 $P(a_n)$ 存储最后一个元素 a_n. 若每台机器内存为 $S=\mathrm{O}(n^{\alpha})$, 该操作可在 $\mathrm{O}(1/\alpha)$ 轮内找到最小值.

● *Broadcast*$(e,P(e),P(c_1),P(c_n))$[6]: 在 MPC 模型中, 将消息 e 从机器 $P(e)$ 广播到包含区间 (c_1,c_n) 内元素的连续机器集. 若每台机器内存为 $S=\mathrm{O}(n^{\alpha})$, 则该操作可在 $\mathrm{O}(1/\alpha)$ 轮内完成. $P(c_1)$ 和 $P(c_n)$ 分别表示存储第 1 个元素 c_1 和最后一个元素 c_n 的机器.

接下来, 提供 4 个用于设计动态 SSSP 算法的 MPC 子程序. 第 1 个子程序是针对 MapReduce 模型设计的快速矩阵乘法算法[40], 如算法 1 所示.

算法 1. *MPC-FMM*(A_1,A_2,C,n,α)[40].

输入: 两个 $n\times n$ 矩阵 A_1 和 A_2, $\mathrm{O}(n^{\omega(2-\alpha)/2})$ 台机器, 每台机器内存为 $\mathrm{O}(n^{\alpha})$ $(\alpha\in(0,1))$, n_0 是快速矩阵乘积算法中计算解矩阵的最小维度 (例如 Strassen 算法中, $n_0=2$);
输出: 解矩阵 $C=A_1\times A_2$.

1. **if** $n\leqslant n^{\omega(2-\alpha)/2}/n_0^{\omega}$ **then**
2. 使用快速矩阵乘法 (计算复杂度为 $\mathrm{O}(n^{\omega})$) 在本地计算 $C=A_1\times A_2$; ▷ 例如: $n_0^{\omega}=7$ 表示 Strassen 算法中每次递归需计算 7 次子矩阵乘法;
3. **else**
4. 将 $n\times n$ 矩阵 A_1 和 A_2 划分为 $(n/n_0)\times(n/n_0)$ 子矩阵 S_i 和 T_i, 并将这些子矩阵重新分配到对应机器上;
5. **for** $i=1,2,\ldots,n_0^{\omega}$ **do**
6. 对所有 i, 并行执行 *MPC-FMM*$(S_i,T_i,Q_i,n/n_0^{\omega},\alpha)\alpha)$;
7. **end**
8. 利用所有子矩阵 Q_i 在本地计算矩阵 C, 其中 $i\in\{1,2,\ldots,n_0^{\omega}\}$;
9. **return** C;
10. **end**

引理 4. 算法 1[40]. 在 MPC 模型中, 每台机器内存大小为 $\mathrm{O}(n^{\alpha})$, 参数 $\alpha\in(0,1)$. 给定两个 $n\times n$ 矩阵 A_1 和 A_2, 存在一个并行算法 *MPC-FMM*(A_1,A_2,C,n,α), 该算法可在 $\mathrm{O}(\log n)$ 轮内使用 $\mathrm{O}(n^{\omega(2-\alpha/2)})$ 台机器计算 A_1 和 A_2 的矩阵乘法, 从而得到矩阵 C. 参数 ω 表示矩阵乘法的算术复杂度为 $\mathrm{O}(n^{\alpha})$.

引理 4 采用快速矩阵乘法 (如 Strassen 矩阵乘法) 来降低总内存需求.

第 2 个子程序[41] (算法 2) 利用半环上矩阵-矩阵乘法来计算所有顶点对之间距离.

算法 2. *MPC-APSP*(A,n,α) [41].

输入: 一个 $n\times n$ 矩阵 A, 分布在 $\mathrm{O}(n^{2-\alpha})$ 台编号 $P_{i,j,1}$ 的机器上, 其中 $i,j\in\{1,2,\ldots,n^{1-\alpha/2}\}$, 每台机器内存为 $\mathrm{O}(n^{\alpha})$ $(\alpha\in(0,1))$;
输出: 距离矩阵 A.

1. **for** $h=1,2,\ldots,\log n$ **do**

2. 机器 $P_{i,j,1}$ $(i,j\in\{1,2,\dots,n^{1-\alpha/2}\})$ 将大小为 $n^{\alpha/2}\times n^{\alpha/2}$ 的子矩阵 $A_{i,j}$ 广播给其他 $O(n^{1-\alpha/2})$ 台编号为 $P_{i,j,k}$ 的机器, $k\in\{2,3,\dots,n^{1-\alpha/2}\}$;

3. 机器 $P_{i,j,1}$ $(i,j\in\{1,2,\dots,n^{1-\alpha/2}\})$ 将子矩阵 $A_{i,j}$ 广播给其他 $O(n^{1-\alpha/2})$ 台编号为 $P_{k,i,j}$ 的机器, $k\in\{2,3,\dots,n^{1-\alpha/2}\}$;

4. 每台机器 $P_{i,j,k}$ $(i,j,k\in\{1,2,\dots,n^{1-\alpha/2}\})$ 计算 $A_{i,k}=A_{i,j}\otimes A_{j,k}$; ▷ $A_{i,j}\otimes A_{j,k}$ 表示半环 $(\min,+)$ 上 $A_{i,j}$ 和 $A_{j,k}$ 的乘法

5. 在编号为 $P_{i,j,k}$ $(j\in\{1,2,\dots,n^{1-\alpha/2}\})$ 的机器之间执行广播操作, 计算子矩阵 $A_{i,k}$ 的部分和, 以得到 $A_{i,k}$, 其中 $i,k\in\{1,2,\dots,n^{1-\alpha/2}\}$;

6. **return** A;

7. **end**

引理 5. 算法 2[41]. 在 MPC 模型中, 每台机器内存大小为 $O(n^{\alpha})$, 内存参数 $\alpha\in(0,1)$. 给定一个图 $G=(V,E)$, 其中 $|V|=n$, 以及对应的邻接矩阵 A, 存在一个并行算法 *MPC-APSP*(A,n,α), 该算法在 $O(\alpha^{-1}\log n)$ 轮内计算所有顶点对之间距离. 总内存需求为 $O(n^{3-\alpha/2})$.

第 3 个子程序是求解 FFT 的 MPC 算法, 所需轮数为 $O(\log n)$, 总内存需求为 $O(n)$. 该算法详细证明如下.

引理 6. *MPC-FFT*(p,q,r,n,α). 对于两个次数至多为 n 的多项式 $p,q\in\mathbb{F}[X]$ 以及内存参数 $\alpha\in(0,1)$, 存在一个 MPC 算法 *MPC-FFT*(p,q,r,n,α), 可在 $O(\log n)$ 轮内计算出多项式 p 和 q 的乘积 r. 所需总内存为 $O(n)$.

证明: 根据顺序 FFT 算法[42], 对于两个次数至多为 n 的多项式乘积, 其次数不大于 $2n$. 取 $N=2^{\lceil\log_2(2n+1)\rceil}$ 并在多项式 $p(X)$ 和 $q(X)$ 后补添高阶系数 0, 至长度为 N. 该操作可在一轮内完成, 且仅需 $O(n)$ 内存.

然后, 使用递归 FFT 算法[42], 第 1 次递归将 N 个系数按奇偶下标分成两个大小为 $N/2$ 的系数向量. 由于每台机器内存为 $O(n^{\alpha})$, 经过 $O((1-\alpha)\log n)$ 次递归后, 可在单台机器上对大小为 $O(n^{\alpha})$ 的系数向量执行 FFT 计算. 每个处理器本地得到 FFT 结果后, 反向递归合并计算得到多项式 $p(X)$ 和 $q(X)$ 的插值结果, 这一过程需 $O(\log n)$ 轮完成.

通过逐点乘积, 可在一轮内得到多项式乘积 $r(X)$ 的插值结果, 大小为 N. 最后, 执行与正向 FFT 类似的操作, 反向计算, 得到多项式乘积 $r(X)$ 的系数向量, 大小不超过 $2n$. 证毕.

在设计并行动态 SSSP 算法时, 引理 6 所述的并行 FFT 算法仅用于两个多项式乘法. 为在单边更新下实现并行动态 SSSP 的低轮数复杂度, 将在第 4.1 节中提出一种新方法, 可在常数轮内完成多项式乘法.

随后, 设计一组基础子程序用于更新 MPC 模型中的动态 SSSP, 详见第 4.3 节. 第 1 个子程序 (算法 3) 如下.

算法 3. *MPC-VVOP*(v_1,v_2,V,n,α).

输入: $O(n^{2-\alpha})$ 台编号 $P_{i,j}$ $(i,j\in\{1,2,\dots,n^{1-\alpha/2}\})$ 的机器; 两个 $1\times n$ 向量 v_1 和 v_2 存储在 $O(n^{1-\alpha})$ 台机器, 这些编号为 $P_{i,1}$ $(i\in\{1,2,\dots,n^{1-\alpha/2}\})$, 每台机器内存大小为 $O(n^{\alpha})$ $(\alpha\in(0,1))$;

输出: 向量 v_1 和 v_2 的外积 V.

1. 机器 $P_{i,1}$ 将向量 v_1 的子向量 v_1^i 广播给其他 $O(n^{1-\alpha/2})$ 台机器, 这些机器编号为 $P_{i,j}$, 其中 $j\in\{2,3,\dots,n^{1-\alpha/2}\}$, $i\in\{1,2,\dots,n^{1-\alpha/2}\}$;

2. 机器 $P_{i,1}$ 将向量 v_2 的子向量 v_2^i 广播给其他 $O(n^{1-\alpha/2})$ 台机器, 这些机器编号为 $P_{j,i}$, 其中 $j\in\{1,2,\dots,n^{1-\alpha/2}\}$, $i\in\{2,3,\dots,n^{1-\alpha/2}\}$;

3. 每台机器 $P_{i,j}$ 计算子向量 v_1^i 和 v_2^j 的外积 $V_{i,j}$ $(i,j\in\{1,2,\dots,n^{1-\alpha/2}\})$;

4. **return** V;

引理 7. 算法 3. 给定两个 $1\times n$ 向量 v_1 和 v_2, 以及内存参数 $\alpha\in(0,1)$, 存在一个并行算法 *MPC-VVOP*(v_1,v_2,V,n,α), 可在 $O(1)$ 轮内计算出向量 v_1 和 v_2 的外积 V (即 $v_1^{\mathrm{T}}\times v_2$). 所需总内存为 $O(n^2)$.

证明: 外积 $v_1^{\mathrm{T}}\times v_2$ 是一个 $n\times n$ 矩阵 V. 由于每台机器内存为 $O(n^{\alpha})$, 存储该矩阵需要 $O(n^{2-\alpha})$ 台机器. 与两个 $n\times n$ 矩阵的乘法不同, 由于矩阵 V 是两个向量外积, 所以 V 中每个元素的计算仅需一个部分和. 因此无需额外机器来降低轮数复杂度.

计算外积时, 将每个向量 (v_1 和 v_2) 划分为大小为 $1\times n^{\alpha/2}$ 的子向量, 并广播至 $O(n^{1-\alpha/2})$ 台其他机器 (此过程需两轮). 这些机器并行计算子向量外积的部分和. 最后, 从所有机器接收部分和, 并将它们整合起来, 即可得到完整外积. 整个过程仅需一轮, 总内存需求为 $O(n^2)$. 证毕.

最后, 给出在半环 $(\min,+)$ 上矩阵与向量相乘的轮数复杂度和总内存 (算法 4), $A\otimes u$ 表示半环 $(\min,+)$ 上矩阵 A 和向量 u 的乘积.

算法 4. *MPC-VM*(A,u,n,α).

输入: 大小为 $n\times n$ 矩阵 A 均匀存储在 $O(n^{2-\alpha})$ 台机器, 这些机器编号 $P_{i,j}$ $(i,j\in\{1,2,\ldots,n^{1-\alpha/2}\})$; 一个 $1\times n$ 向量 u 存储在 $O(n^{1-\alpha})$ 台机器, 这些机器编号为 $P_{i,1}$ $(i\in\{1,2,\ldots,n^{1-\alpha/2}\})$, 每台机器内存大小为 $O(n^{\alpha})$ $(\alpha\in(0,1))$;
输出: $A\otimes u$.

1. 机器 $P_{i,1}$ 将向量 u 的子向量 u_i 广播给其他 $O(n^{1-\alpha/2})$ 台机器, 这些机器编号为 $P_{j,1}$, 其中 $i,j\in\{1,2,\ldots,n^{1-\alpha/2}\}$;
2. 每台机器 $P_{i,j}$ 本地计算子矩阵 $A_{i,j}$ 和子向量 u_j 的乘积, $i,j\in\{1,2,\ldots,n^{1-\alpha/2}\}$;
3. 对编号为 $P_{i,j}$ 的机器执行广播操作, 并计算子向量 $(A\otimes u)_{i,j}$ 的部分和, 得到 $(A\otimes u)_i$, 其中 $i,j\in\{1,2,\ldots,n^{1-\alpha/2}\}$;

引理 8. 算法 4. 在 MPC 模型中, 每台机器内存大为 $O(n^{\alpha})$ 且给定参数 $\alpha\in(0,1)$, 存在一个并行算法 *MPC-VM*(A,u,n,α), 可在 $O(1)$ 轮内计算半环 $(\min,+)$ 上 $1\times n$ 向量 u 和 $n\times n$ 矩阵 A 乘积. 所需总内存为 $O(n^2)$.

证明: 由于矩阵 A 的大小为 $n\times n$, 且每台机器内存为 $O(n^{\alpha})$, 因此需要 $O(n^{2-\alpha})$ 台机器来存储矩阵 A. 对于向量 a, 将其划分为 $n^{1-\alpha/2}$ 个子向量, 每个子向量包含 $O(n^{\alpha})$ 个元素, 并发送到指定的 $n^{1-\alpha/2}$ 台机器. 这 $n^{1-\alpha/2}$ 台机器存储矩阵 A 的不同子矩阵. 此操作可在一轮内执行半环上加法操作. 随后, 对于最终解向量 (总共 n 个元素) 中的每个元素, 会得到 $n^{1-\alpha/2}$ 个待比较元素. 若 $\alpha\in(0,2/3)$, 则每个元素的 $n^{1-\alpha/2}$ 个待比较元素无法全部存入单台机器. 此时, 调用第 3.3 节广播操作, 可在 $O((1-\alpha)/\alpha)$ 轮内对最终解向量中每个元素进行比较. 若 $\alpha\in[2/3,1)$, 则仅需一轮即可获得最终结果. 证毕.

4 MPC 模型中完全动态 SSSP 算法

我们针对 MPC 模型中完全动态精确 SSSP 问题, 提出一种高效的并行算法. 该算法包含两个部分: 预处理算法和单边更新算法. 具体而言, 第 4.1 节提出一个可在常数轮内完成的高效多项式乘积算法; 第 4.2 节介绍所设计的并行预处理算法; 第 4.3 节具体描述在单条边插入、删除及权重改变时的并行更新算法.

4.1 一个用于多项式乘法的常数轮并行算法

在详述并行动态 SSSP 算法之前, 首先介绍一种用于多项式乘法的高效算法. 该算法可在常数轮内计算次数至多为 n 的两个多项式乘积, 其轮数复杂度优于 MPC 模型中的 FFT 算法.

引理 9. *MPC-PM*$(p,q,r,n,\alpha,\mathbb{F}[X]/\langle X^n\rangle)$. 给定两个多项式 $p,q\in\mathbb{F}[X]/\langle X^n\rangle$, 其次数至多为 n, 内存参数 $\alpha\in(0,1)$, 存在一个 MPC 算法 *MPC-PM*$(p,q,r,n,\alpha,\mathbb{F}[X]/\langle X^n\rangle)$, 可在 $O(\alpha^{-1})$ 轮内计算出多项式 p,q 的乘积, 所需总内存为 $O(n^2)$.

证明: 已知 FFT 的计算复杂度为 $O(n\log n)$ [42]. 本文提出一种常数轮的多项式乘法算法, 该算法基于标准多项式乘法, 其算术复杂度为 $O(n^2)$.

设 $p,q\in\mathbb{F}[X]/\langle X^n\rangle$, 其中 $p=a_0+a_1X+a_2X^2+\ldots+a_{n-1}X^{n-1}$ 且 $q=b_0+b_1X+b_2X^2+\ldots+b_{n-1}X^{n-1}$. 则乘积 $r=p\cdot q$ 亦属于 $\mathbb{F}[X]/\langle X^n\rangle$. 定义系数向量 $a=(a_0,a_1,a_2,\ldots,a_{n-1})$ 和 $b=(b_0,b_1,b_2,\ldots,b_{n-1})$. 利用计算复杂度为 $O(n^2)$ 的直接算法, 仅需计算如下向量 a 和 b 的外积 R:

$$R=\begin{bmatrix} r_{0,0} & r_{0,1} & \cdots & r_{0,n-1}\\ r_{1,0} & r_{1,1} & \cdots & r_{1,n-1}\\ \vdots & \vdots & \ddots & \vdots\\ r_{n-1,0} & r_{n-1,1} & \cdots & r_{n-1,n-1}\end{bmatrix}=a^{\mathrm{T}}b=\begin{bmatrix} a_0b_0 & a_0b_1 & \cdots & a_0b_{n-1}\\ a_1b_0 & a_1b_1 & \cdots & a_1b_{n-1}\\ \vdots & \vdots & \ddots & \vdots\\ a_{n-1}b_0 & a_{n-1}b_1 & \cdots & a_{n-1}b_{n-1}\end{bmatrix}.$$

由上式可知, 每个元素 $r_{i,j}$, $i\in\{0,1,\ldots,n-1\}$, $j\in\{0,1,\ldots,n-1\}$ 的下标之和即为变量 X 的次数. 同理, 元素 a_i 和 b_j 的下标之和也表示变量 X 的次数, 其中 $i\in\{0,1,\ldots,n-1\}$, $j\in\{0,1,\ldots,n-1\}$. 然而, 由于 $r\in\mathbb{F}[X]/\langle X^n\rangle$, 对于 X 次数高于 $n-1$ 的元素项, 需通过模 X^n 进行简化.

为计算变量 X^h 的系数, 其中 $h\in[0,n)$, 且 h 为整数, 在得到 $r_{i,j}$ 的下标和后, 需减去 h. 基于此观察, 通过对矩阵 R 和 $a^{\mathrm{T}}b$ 中元素进行适当修改, 可得到修改后矩阵 R':

$$R'=\begin{bmatrix} r_{0,0} & r_{1,n-1} & \cdots & r_{n-1,1}\\ r_{0,1} & r_{1,0} & \cdots & r_{n-1,2}\\ \vdots & \vdots & \ddots & \vdots\\ r_{0,n-1} & r_{1,n-2} & \cdots & r_{n-1,0}\end{bmatrix}=\begin{bmatrix} b_0a_0 & b_1a_{n-1} & \cdots & b_{n-1}a_1\\ b_0a_1 & b_1a_0 & \cdots & b_{n-1}a_2\\ \vdots & \vdots & \ddots & \vdots\\ b_0a_{n-1} & b_1a_{n-2} & \cdots & b_{n-1}a_0\end{bmatrix}.$$

对矩阵 R 中的每一行元素求和, 即可得到多项式 r 的系数向量 $c=(c_0,c_1,c_2,\ldots,c_{n-1})$. 因此, 只需计算如下向量矩阵乘法:

$$c=Ab^{\mathrm{T}}=\begin{bmatrix} a_0 & a_{n-1} & \cdots & a_1\\ a_1 & a_0 & \cdots & a_2\\ \vdots & \vdots & \ddots & \vdots\\ a_{n-1} & a_{n-2} & \cdots & a_0\end{bmatrix}\cdot\begin{bmatrix} b_0\\ b_1\\ \vdots\\ b_{n-1}\end{bmatrix}.$$

至此, $a^{\mathrm{T}}b$ 的计算已转化为上述向量-矩阵乘法. 为此, 将矩阵 A 划分为 $n^{2-\alpha}$ 个子矩阵, 每个子矩阵大小为 $n^{\alpha/2}\times n^{\alpha/2}$. 由于每台机器内存大小为 $\mathrm{O}(n^{\alpha})$, 计算 Ab^{T} 需要 $n^{2-\alpha}$ 台机器, 总内存需求为 $\mathrm{O}(n^2)$. 随后, 将向量 b^{T} 划分为 $\mathrm{O}(n^{1-\alpha/2})$ 个子向量, 每个子向量包含 $n^{\alpha/2}$ 个元素, 并在常数轮内将其与矩阵 A 的对应子矩阵一起分发到 $n^{1-\alpha/2}$ 台机器. 之后, 在每台机器上执行本地计算得到矩阵 R'.

然而, 各处理器本地计算后, 矩阵 R' 的元素仍为部分和元素. 因此, 需为矩阵 R' 中每个元素计算剩余的 $n^{1-\alpha/2}$ 个部分和. 当 $\alpha\in(0,2/3)$ 时, 这 $n^{1-\alpha/2}$ 个元素无法存入单台机器. 为解决此问题, 可采用第 3.3 节的广播操作, 在 $\mathrm{O}((1-2\alpha)/\alpha)$ 轮内计算矩阵 R' 中每个元素的剩余部分和. 另一方面, 当 $\alpha\in[2/3,1)$ 时, 则仅需一轮即可得到最终结果.

综上分析, 在域 $\mathbb{F}[X]/\langle X^n\rangle$ 上计算两个多项式乘法所需轮数复杂度为常数, 该常数与参数 α 有关. 证毕.

4.2 并行预处理算法

本文针对 MPC 模型中动态精确 SSSP 问题, 设计了一个有效的并行预处理算法, 用于构建高效数据结构, 以便在第 4.3 节中更新源点到其他顶点的距离. 初始时, 对原始图 G 进行预处理, 获得一种能处理边更新引起距离变化的数据结构. 为便于说明, 先在无权图 $G=(V,E)$ 上验证该预处理算法的有效性.

引理 1 表明, 通过多项式矩阵求逆方法计算的所有距离均为整数. 此外, 多项式矩阵逆中变量元素的最大次数决定了有向图 G 中顶点之间最大距离. 在无权图中, 参数 d 表示两个顶点之间至多包含 d 条边的最大距离, 因为多项式矩阵逆的每个元素都属于 $\mathbb{F}[X]/\langle X^n\rangle$. 考虑到在任何两个顶点之间路径上至多包含 $n-1$ 条边, 可设置 $d=n^s$, 其中 $s\in(0,1)$, 以确保 d 取值适当. 因此, 计算多项式矩阵逆即可求解顶点之间 $\leqslant n^s$ 条边的最短路径长度问题. 算法 5 给出了预处理算法的执行过程.

算法 5. *MPC*(n^{α}) 模型中动态 SSSP 预处理算法.

输入: 给定一个图 $G=(V,E,W)$, 有限域 $\mathbb{F}[X]/\langle X^{n^s}\rangle$ ($s\in(0,1)$), 一个 $n\times n$ 空矩阵 B;
输出: 多项式矩阵 M^{-1}.

1. 第 1 部分: 预处理图的边集和顶点集; ▷ 参考引理 1
2. 对于每个顶点 $u\in V$, 设置 $B_{u,u}=a_{u,u}X$, 其中 $a_{u,u}$ 是从 $\mathbb{F}$ 中随机均匀抽样的一个元素;
3. **if** $(u,v)\in E$ **then**
4. $\quad B_{u,v}=a_{u,v}X^{w_{u,v}}$, 其中 $a_{u,v}$ 是从 $\mathbb{F}$ 中随机均匀抽样的一个元素;
5. **else**

6. $B_{u,v}=0$;
7. **end**
8. 第 2 部分: 计算至多有 n^s 条边的多项式矩阵 M^{-1};
9. 计算 $M=\mathbb{I}+B$, 其中 $\mathbb{I}$ 是 $n\times n$ 单位矩阵;
10. **for** $i=1$ to $\lceil\log n\rceil$ **do**
11. 在进行矩阵-矩阵乘法时, 执行子程序 $MPC\text{-}FMM(B^{2^{i-1}},B^{2^{i-1}},B^{2^i},n,\alpha)$, 并在进行多项式元素乘法时执行子程序 $MPC\text{-}FFT(B^{2^{i-1}}_{h,k},B^{2^{i-1}}_{k,j},B^{2^i}_{h,j},n^s,\alpha)$, 其中 $h,k,j\in\{1,2,\ldots,n\}$; ▷ 参考引理 4 和引理 6, 计算 $B^{2^i}=B^{2^{i-1}}\times B^{2^{i-1}}$
12. 在进行矩阵-矩阵乘法时, 执行子程序 $MPC\text{-}FMM(M^{-1},\mathbb{I}+B^{2^i},M^{-1},n,\alpha)$, 并在进行多项式元素乘法时执行子程序 $MPC\text{-}FFT(M^{-1}_{h,k},(\mathbb{I}+B^{2^i})_{k,j},M^{-1}_{h,j},n^s,\alpha)$, 其中 $h,k,j\in\{1,2,\ldots,n\}$; ▷ 参考引理 4 和引理 6, 计算 $M^{-1}=M^{-1}\times(\mathbb{I}+B^{2^i})$
13. **end**

算法 5 由两部分组成. 第 1 部分使用引理 1 将图 G 的顶点集和边集转换为多项式矩阵 B (如算法 5 第 2–7 行所示). 第 2 部分使用两个子程序: MPC 模型中矩阵乘积算法和 FFT 算法, 计算域 $(\mathbb{F}[X]/\langle X^{n^s}\rangle)^{n\times n}$ 上的多项式矩阵逆 M^{-1} (如算法 5 第 9–13 行所示).

复杂度分析: 根据引理 1, 有以下结果.

引理 10. 算法 5 的轮数复杂度. 给定一个无权图 $G=(V,E)$, 其中 $|V|=n$. 对于 MPC 模型, 每台机器内存大小为 $O(n^\alpha)$ 且参数 $\alpha\in(0,1)$, 设 $M=(\mathbb{I}-B)\in(\mathbb{F}[X]/\langle X^{n^s}\rangle)^{n\times n}$, 则多项式矩阵逆 $M^{-1}=\prod_{k=0}^{\log n^s}(\mathbb{I}+B^{2^k})$ 可通过算法 5 在 $O(\log^3 n)$ 轮内计算, 且该计算以高概率成功. 参数 s 是一个待确定值, 取值在 $(0,1)$ 之间. 算法 5 所需总内存为 $O(n^{\omega+s-(\omega/2-1)\alpha})$.

证明: 算法 5 第 1 部分可以在常数轮内完成. 下面重点分析第 2 部分. 在算法 5 第 9 行, 计算 $M=\mathbb{I}+B$ 可通过复制多项式矩阵 B 并对其对角元素加 1 实现. 利用第 3.3 节广播操作, 将元素 1 广播到存储矩阵 B 对角线元素的机器上, 仅需 $O(\alpha^{-1})$ 轮.

由于多项式矩阵逆 $M^{-1}=\prod_{k=0}^{\log n^s}(\mathbb{I}+B^{2^k})$, 使用引理 4 在 $O(\log n)$ 轮内计算矩阵 B^2, 所需总内存为 $O(n^{\omega(2-\alpha)/2+\alpha})$. 结合路径加倍策略, 可在 $O(\log^2 n)$ 轮内获得 B^n 和所有 B^{2k}, 其中 $k\in\{0,1,\ldots,\log n\}$ (算法 5 第 11 行).

由于矩阵 B 中每个元素都是次数至多为 n^s 的多项式, 利用引理 6, 矩阵 B 中两个元素乘积的计算在 $O(\log n^s)$ 轮内完成, 并占用 $O(n^s)$ 内存 (算法 5 第 11 行). 此外, 由于 $\prod_{k=0}^{\log n^s}(I+B^{2^k})$ 的结构 (算法 5 第 10 行) , 对次数至多为 n^s 的多项式矩阵 M 进行预处理需要 $O(\log^3 n)$ 轮. 所需总内存为 $O(n^{\omega+s-(\omega/2-1)\alpha})$. 由引理 1 可知, 算法 5 能以高概率成功计算多项式矩阵逆 M^{-1}. 证毕.

4.3 并行单边更新算法

本文主要考虑动态图的单边更新, 包括边的插入与删除以及边权重的改变. 每次边更新后, 图 G 邻接矩阵 A 中的一个元素发生变化, 导致由引理 1 构建的多项式矩阵 $B(G)$ 也随之改变. 我们使用域 $\mathbb{F}[X]$ 上多项式 uv^{T} 来表示多项式矩阵 M 的每次更新, 其中 $u,v\in(\mathbb{F}[X]/\langle X^{n^s}\rangle)^n$ 为两个多项式向量. 算法 6 描述了 MPC 模型中求解动态 SSSP 问题的单边更新算法.

算法 6. $MPC(n^\alpha)$ 模型中动态 SSSP 单边更新算法.

输入: 给定一个图 $G=(V,E,W)$, 源点 s, 预处理算法 (算法 5) 计算输出的多项式矩阵逆 M^{-1}, 两个 $1\times n$ 向量 $u,v\in(\mathbb{F}[X]/\langle X^{n^s}\rangle)^n(s\in(0,1))$, 每个向量只有一个非零元, 表示相应边 (i,j) 的更新;
输出: 多项式矩阵 M^{-1}, 距离矩阵 D, 从源顶点 s 到采样顶点的距离向量 D_s.

1. 第 1 部分: 更新路径包含至多 n^s 条边的多项式矩阵逆 M^{-1}; ▷ 更新短边距离

2. 执行 $Broadcast(u_i^{\mathrm{T}}, P(u_i^{\mathrm{T}}), P(M_{1,i}^{-1}), P(M_{n,i}^{-1}))$;
3. 执行子程序 $MPC\text{-}PM(M_{k,i}^{-1}, u_i^{\mathrm{T}}, \tilde{u}_k^{\mathrm{T}}, n^s, \alpha, \mathbb{F}[X]/\langle X^{n^s}\rangle)$, 其中 $k \in \{1,2,\ldots,n\}$; ▷ 参考引理 9, 计算 $\tilde{u}^{\mathrm{T}} = M^{-1}u^{\mathrm{T}}$
4. 执行 $Broadcast(v_j, P(v_j), P(M_{j,1}^{-1}), P(M_{j,n}^{-1}))$;
5. 执行子程序 $MPC\text{-}PM(M_{j,k}^{-1}, v_j, \tilde{v}_k, n^s, \alpha, \mathbb{F}[X]/\langle X^{n^s}\rangle)$, 其中 $k \in \{1,2,\ldots,n\}$; ▷ 参考引理 9, 计算 $\tilde{v} = vM^{-1}$
6. 执行 $MPC\text{-}PM(v_j, u_j^{\mathrm{T}}, p, n^s, \alpha, \mathbb{F}[X]/\langle X^{n^s}\rangle)$; ▷ 参考引理 9, 计算 $p = v_j \cdot u_j^{\mathrm{T}}$
7. 计算 $q = 1 - p$;
8. **for** $i = 1$ to $\lceil \log n^s \rceil$ **do** ▷ 计算 $(1 + v\tilde{u}^{\mathrm{T}})^{-1}$
9. 执行子程序 $MPC\text{-}PM(p, p, p, n^s, \alpha, \mathbb{F}[X]/\langle X^{n^s}\rangle)$; ▷ 参考引理 9
10. 执行子程序 $MPC\text{-}PM(q, (1+p), q, n^s, \alpha, \mathbb{F}[X]/\langle X^{n^s}\rangle)$; ▷ 参考引理 9
11. **end**
12. 执行 $Broadcast(q, P(q), P(\tilde{v}_1), P(\tilde{v}_n))$; ▷ 参考第 3.3 节广播操作
13. 执行 $MPC\text{-}PM(q, \tilde{v}_i, \tilde{v}_i, n^s, \alpha, \mathbb{F}[X]/\langle X^{n^s}\rangle)$; ▷ 参考引理 9, 计算 $\tilde{v} = (1 + v\tilde{u}^{\mathrm{T}})^{-1}\tilde{v}$
14. 执行子程序 $MPC\text{-}VVOP(\tilde{u}^{\mathrm{T}}, \tilde{v}, V, n, \alpha)$ 并执行子程序 $MPC\text{-}FFT(\tilde{u}_i^{\mathrm{T}}, \tilde{v}_j, V_{i,j}, n^s, \alpha)$ 来计算两个向量外积中两个多项式元素乘法; ▷ 参考引理 7, 计算 $V = \tilde{u}^{\mathrm{T}} \times \tilde{v}$
15. 计算 $M^{-1} = M^{-1} - V$;
16. 执行 $Find\text{-}Minimum(M_{i,j}^{-1}, P((M_{i,j}^{-1})^{[0]}), P((M_{i,j}^{-1})^{[n^s]}))$, 在存储 M^{-1} 的每个多项式元素系数的机器中, 找到使 $(M_{i,j}^{-1})^{[k]} \neq 0$ 的最小 $k \in \{1,2,\ldots,n^s\}$, 以获得每条路径至多 n^s 条边的 $n \times n$ 距离矩阵 D, 其中 $i, j \in \{1,2,\ldots,n\}$; ▷ 参考第 3.3 节 *Find-Minimum* 算法
17. 第 2 部分: 更新路径包含至少 $n^s + 1$ 条边的多项式矩阵 M^{-1}; ▷ 更新长边距离
18. 从给定顶点集 V 中随机均匀采样顶点, 采样概率为 n^{-s}, 以得到一个顶点子集, 包含 $O(n^{1-s}\ln n)$ 采样顶点; ▷ 参考引理 3
19. 查询距离矩阵 D 中这些采样顶点对应的行和列, 并得到一个大小为 $(n^{1-s}\ln n) \times (n^{1-s}\ln n)$ 的距离子矩阵 H;
20. 执行 MPC-APSP$(H, n^{1-s}\ln n, \alpha)$ 以获得采样顶点的距离矩阵 H; ▷ 参考引理 5
21. 查询距离矩阵 D 以获得从源顶点 s 到这些采样顶点经过 $\leqslant n^s$ 条边的最短路径长度, 并将它们组合成一个 $1 \times n$ 向量 D_s;
22. 执行子程序 MPC-VM$(H^{\mathrm{T}}, D_s^{\mathrm{T}}, n^{1-s}\ln n, \alpha)$ 以获得从源顶点 s 到这些采样顶点的距离向量 D_s^{T}; ▷ 参考引理 8

算法 6 由两部分组成. 第 1 部分是计算短边最短路径长度 (算法 6 第 1–16 行). 第 2 部分是计算长边最短路径长度并比较得到最终距离 (算法 6 第 17–22 行).

根据引理 2, 每次边更新仅需计算 $M^{-1}u^{\mathrm{T}}(1 + vM^{-1}u^{\mathrm{T}})^{-1}vM^{-1}$, 这对应于算法 6 第 2–15 行. 为与文献 [12] 的研究设置保持一致, 设 $\tilde{u}^{\mathrm{T}} = M^{-1}u^{\mathrm{T}}$, $\tilde{v} = (1 + vM^{-1}u^{\mathrm{T}})^{-1}vM^{-1} = (1 + v\tilde{u}^{\mathrm{T}})^{-1}vM^{-1}$. 然后, 每次边更新后得到两个多项式向量 u, v, 且只需计算多项式向量 $\tilde{u}^{\mathrm{T}}$ (算法 6 第 2、3 行) 和 $\tilde{v}$ (算法 6 第 4–13 行). 最后计算多项式向量 $\tilde{u}^{\mathrm{T}}$ 和 $\tilde{v}$ 的外积 (算法 6 第 14 行).

更新顶点之间短边最短路径长度后, 可得到一个路径上至多包含 n^s 条边的距离矩阵 D (算法 6 第 16 行). 使用该距离矩阵 D 更新采样顶点的距离子矩阵 H (算法 6 第 18–20 行). 采用路径分解方法, 将长边路径划分为多条短边路径, 从而可在两步内计算从源顶点 s 到其他顶点的长边距离.

具体而言, 将长边路径分解为 3 条短边路径. 第 1 步计算和更新前两条短边最短路径长度 (算法 6 第 21–22 行). 第 2 步在查询从源顶点 s 到任何其他顶点的距离时执行: 首先将第 3 条短边最短路径长度与运行算法 6 得到的短边最短路径长度结合, 得到从源顶点 s 到任意其他顶点的长边最短路径长度; 然后将其与源顶点到该顶点的短边距离进行比较, 得到最终结果. 更多细节详见第 4.4 节定理 1 的证明.

复杂度分析: 接下来分析所设计并行更新算法 (即算法 6) 的轮数复杂度和总内存开销.

引理 11. 算法 6 的轮数复杂度. 给定无权图 $G=(V,E,)$, 其中 $|V|=n$. 在 MPC 模型中, 每台机器内存大小为 $O(n^{\alpha})$ 且参数 $\alpha\in(0,1)$, 存在一个并行动态 SSSP 算法 (算法 6) , 可在 $O(\alpha^{-1}\log n)$ 轮内更新源点到其他顶点的距离且以高概率成功发生. 此外, 算法 6 所需总内存为 $O(n^{2+s}+(n^{1-s}\ln n)^{3-\alpha/2})$.

证明: 引理 11 的证明包括两部分. 首先证明算法 6 第 1 部分 (算法 6 第 1–16 行) 所需轮数复杂度和总内存.

如前所述, $\tilde{u}^{\mathrm{T}}=M^{-1}u^{\mathrm{T}}$ 和 $\tilde{v}=(1+v\tilde{u}^{\mathrm{T}})^{-1}vM^{-1}$. 执行算法 5 后, 得到 M^{-1}, 而 u^{T} 是一个多项式向量, 仅含一个非零多项式元素. 因此, 可利用引理 9 计算 u^{T} 的非零元素与多项式矩阵 M^{-1} 对应列的乘积. 由于至多有 n 个多项式元素, 其阶数至多为 n^{s}, 所需总内存为 $O(n^{2s})$. 使用广播算法[6]将这些多项式元素发送到指定机器进行本地计算. 利用引理 9 获得向量 $\tilde{u}^{\mathrm{T}}$, 可在 $O(\alpha^{-1})$ 轮内完成.

为计算多项式向量 $\tilde{v}$, 首先需计算 $(1+v\tilde{u}^{\mathrm{T}})^{-1}$, 这是域 $\mathbb{F}[X]/\langle X^{n^s}\rangle$ 上多项式. 使用引理 9 和 $p=\prod_{k=0}^{\log n^s}(1+(-u^{\mathrm{T}}\tilde{v})^{2^k})$, 可在 $O(\alpha^{-1}\log n^s)$ 轮内以 $O(n^{2s})$ 总内存计算 $(1+v\tilde{u}^{\mathrm{T}})^{-1}$.

为计算 vM^{-1}, 可采用与计算 $\tilde{u}^{\mathrm{T}}$ 类似的方法, 仅需 $O(\alpha^{-1})$ 轮完成. 由于 vM^{-1} 是一个多项式向量, 而 $(1+v\tilde{u}^{\mathrm{T}})^{-1}$ 是一个多项式, 将 $(1+v\tilde{u}^{\mathrm{T}})^{-1}$ 广播到存储向量 vM^{-1} 的机器, 仅需 $O(\alpha^{-1})$. 接下来, 执行向量 $\tilde{u}^{\mathrm{T}}$ 和 $\tilde{v}$ 的外积. 根据引理 7, 计算任意两个向量外积的轮数复杂度为 $O(1)$, 所需总内存为 $O(n^2)$. 由于向量 $\tilde{u}^{\mathrm{T}}$ 和 $\tilde{v}$ 的元素是域 $\mathbb{F}[X]/\langle X^{n^s}\rangle$ 上多项式, 可用引理 6 在 $O(\log n^s)$ 轮内以 $O(n^{2+s})$ 总内存计算向量 $\tilde{u}^{\mathrm{T}}$ 和 $\tilde{v}$ 的外积.

对于算法 6 第 2 部分 (算法 6 第 17–22 行), 采样顶点和查询距离矩阵 D 以获得子矩阵 H, 这些操作可在常数轮内完成. 利用引理 5, 执行子程序 $MPC\text{-}APSP(H,n^{1-s}\log n,\alpha)$ (对应于算法 6 第 20 行) 的轮数复杂度为 $O(\alpha^{-1}\log(n^{1-s}\log n))$, 所需总内存为 $O((n^{1-s}\ln n)^{3-\alpha/2})$. 子程序 $MPC\text{-}VM(H^{\mathrm{T}},D_s,n^{1-s}\log n,\alpha)$ 的轮数复杂度为 $O(1)$, 总内存为 $O((n^{1-s}\log n)^2)$.

由于算法 6 的输入结果依赖于算法 5, 因此算法 6 是一个随机算法. 证毕.

类似于文献 [12] 和 Sankowski[14]的研究, 所提并行更新算法需基于采样结果的查询距离矩阵 D. 在高概率下, 该过程不会泄露任何敏感数据. 因此, 所提并行动态更新算法能够正确维护所有最短路径, 即使面对非遗忘性对手 (即对手可以根据历史输出适应性地选择更新操作).

为便于理解引理 1 在算法 5 及引理 2 在算法 6 中的应用, 本文给出实例, 以说明利用多项式矩阵求逆计算和更新短边距离的过程.

例 1: 考虑有向无权图 $G_0=(V_0,E_0)$ 及其邻接矩阵 A_0, 其中 $V_0=\{1,2,3,4\}$.

$$A_0=\begin{bmatrix}0&1&1&1\\1&0&\infty&1\\\infty&1&0&1\\1&\infty&1&0\end{bmatrix}.$$

Step 1: 根据引理 1 构造图 G_0 的多项式矩阵 $B(G_0)\in(\mathbb{F}[X]/\langle X^4\rangle)^{4\times4}$. 由于只有 4 个顶点, 设置 $d=n=4$, 以说明多项式矩阵逆计算顶点间距离的过程, 以及引理 2 如何反映单边更新对顶点之间距离的影响. 由引理 1 可知:

$$B(G_0)=\begin{bmatrix}a_{1,1}X&a_{1,2}X&a_{1,3}X&a_{1,4}X\\a_{2,1}X&a_{2,2}X&0&a_{2,4}X\\0&a_{3,2}X&a_{3,3}X&a_{3,4}X\\a_{4,1}X&0&a_{4,3}X&a_{4,4}X\end{bmatrix}=\begin{bmatrix}X&2X&X&3X\\3X&2X&0&X\\0&3X&4X&2X\\X&0&X&0\end{bmatrix},$$

则有:

$$M=\mathbb{I}-B(G)=\begin{bmatrix}1-X&1-2X&1-X&1-3X\\1-3X&1-2X&1&1-X\\1&1-3X&1-4X&1-2X\\1-X&1&1-X&1\end{bmatrix}.$$

Step 2: 计算 $M^{-1}=(\mathbb{I}-B(G))^{-1}=\sum_{k=0}^{3}B(G_0)^k$, 则有:

$$M^{-1}=\begin{bmatrix} X^0+X^1+10X^2+44X^3 & 2X^1+9X^2+62X^3 & X^1+8X^2+49X^3 & 3X^1+7X^2+55X^3 \\ 3X^1+9X^2+50X^3 & X^0+2X^1+10X^2+50X^3 & 4X^2+36X^3 & X^1+11X^2+45X^3 \\ 11X^2+76X^3 & 3X^1+18X^2+112X^3 & X^0+4X^1+18X^2+94X^3 & 2X^1+11X^2+87X^3 \\ X^1+X^2+21X^3 & 5X^2+27X^3 & X^1+5X^2+26X^3 & X^0+5X^2+18X^3 \end{bmatrix}.$$

由算法 6 第 16 行可知, $G_0=(V_0,E_0)$ 的距离矩阵为:

$$D_0=\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix}.$$

Step 3: 考虑插入边 $(4,1)$, 得到新的邻接矩阵:

$$A_0^{new}=\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & \infty & 1 \\ \infty & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

根据引理 2 以及算法 6, 将新插入边 $(4,1)$ 用两个多项式向量表示:

$$u=\begin{bmatrix} 0 & 0 & 0 & X \end{bmatrix},\ v=\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}.$$

Step 4: 根据引理 2 可知, 仅需计算扰动因子 $M^{-1}u^{\mathrm{T}}(1+vM^{-1}u^{\mathrm{T}})^{-1}vM^{-1}$:

$$M^{-1}u^{\mathrm{T}}(1+vM^{-1}u^{\mathrm{T}})^{-1}vM^{-1}=\begin{bmatrix} -9X^3 & -3X^2-13X^3 & -2X^3 & -3X^3 \\ -X^2-4X^3 & -X^2-15X^3 & -X^3 & -4X^3 \\ -6X^3 & -2X^2-15X^3 & 0 & -2X^3 \\ -3X^2-10X^3 & -X^1-2X^2-16X^3 & -4X^3 & -X^2-11X^3 \end{bmatrix}.$$

则有:

$$(M+u^{\mathrm{T}}v)^{-1}=\begin{bmatrix} X^0+X^1+10X^2+53X^3 & 2X^1+12X^2+75X^3 & X^1+8X^2+51X^3 & 3X^1+7X^2+58X^3 \\ 3X^1+10X^2+54X^3 & X^0+2X^1+11X^2+65X^3 & 4X^2+37X^3 & X^1+11X^2+49X^3 \\ 11X^2+82X^3 & 3X^1+20X^2+127X^3 & X^0+4X^1+18X^2+94X^3 & 2X^1+11X^2+89X^3 \\ X^1+4X^2+31X^3 & X^1+7X^2+43X^3 & X^1+5X^2+30X^3 & X^0+6X^2+29X^3 \end{bmatrix}.$$

由此可知, 新的距离矩阵为:

$$D_0^{\mathrm{new}}=\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

4.4 主要结果

我们提出一种具有低轮数复杂度的高效并行算法, 用于维护给定图中从源顶点到其他顶点的距离. 主要结果如下.

定理 1. 给定一个无向/有向图 $G=(V,E)$, 其中边权重为整数, $|V|=n$, 在单边插入、删除或单边权重改变的更新操作下. 最大边权重和最小边权重的比值为 $\lceil poly(\log n)\rceil$. 在 MPC 模型中, 每台机器内存大小为 $\mathrm{O}(n^{\alpha})$, 参数 $\alpha\in(0,1)$, 存在一个并行算法, 以高概率在 $\mathrm{O}(\log^3 n)$ 轮内预处理图 G, 并在 $\mathrm{O}(\alpha^{-1}\log n)$ 轮内更新从源点 $s\in V$ 到其他 n 个顶点的距离. 查询源点到任意其他顶点之间距离需 $\mathrm{O}(\alpha^{-1})$ 轮. 所需总内存为 $\tilde{\mathrm{O}}(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})$, 当 $\alpha\in(0,1)$ 且 $\omega\simeq 2.3728$ 时, 该值在 $\tilde{\mathrm{O}}(n^{2.276})$ 到 $\tilde{\mathrm{O}}(n^{2.5296})$ 之间波动.

证明: 首先证明无权有向图的情况 (无权无向图类似). 因为任意两个顶点之间最短路径长度所含边数至多为 $n-1$, 所以图 G 中两个顶点的距离不超过 $n-1$. 根据算法 5 (预处理算法) 和引理 10, 可预处理多项式矩阵 M, 其次数至多为 n^s, 并得到多项式矩阵的逆 M^{-1}. 所需轮数复杂度为 $\mathrm{O}(\log^3 n)$, 总内存需求为 $\mathrm{O}(n^{\omega+s-(\omega/2-1)\alpha})$.

每次边更新后, 执行算法 6 (单边更新算法) 来更新次数至多为 n^s 的多项式矩阵逆 M^{-1}. 并作为下次边更新的数据结构. 同时使用算法 6 维护每条路径上至多 n^s 条边的距离矩阵 D 和从源点 s 到采样顶点的距离向量 D_s.

查询从源点 s 到任意顶点 $t\in V$ 的距离时, 首先获得一个 $(n^{1-s}\ln n)\times 1$ 距离向量 D_t, 表示从采样顶点到顶点 t 的

距离. 这可通过查询距离矩阵 D 在常数轮内完成. 然后计算距离向量 D_s 和 D_t 的内积, 以获得从 s 到 t 经过 $\geqslant n^s$ 条边的最短路径长度 (其路径至少包含 n^s 条边). 该计算仅需 $O(\alpha^{-1})$ 轮. 最后, 查询距离矩阵 D, 获得从 s 到 t 经过 $\leqslant n^s$ 条边的 (短边) 最短路径长度, 并将其与 $\geqslant n^s$ 条边的最短路径长度 (长边距离) 进行比较. 较小者即为所求距离.

主要开销来自算法 6. 当 $s=(3-\omega)(1-\alpha/2)/(4-\alpha/2)$ 时, 根据引理 10 和引理 11, 算法 5 和算法 6 所需总内存为 $O(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})$. 注意, 若假设 $\omega\simeq 2.3728$, 由于参数 $\alpha\in(0,1)$, 总内存取值范围为 $O(n^{2.276})$ 到 $O(n^{2.5296})$.

对于加权有向图, 最大边权重与最小边权重的比率为 $\lceil poly(\log n)\rceil$. 通过将边权重除以最小边权值, 可将边权重缩放至区间 $[\lceil poly(\log n)\rceil]$. 对于具有 n 个顶点的图 G, 任意两个顶点之间最大距离不超过 $\lceil poly(\log n)\rceil\cdot n$. 因此, 仅需将模 X^d 的指数 d 调整为 $\lceil poly(\log n)\rceil\cdot n^s$ ($s=(3-\omega)(1-\alpha/2)/(4-\alpha/2)$). 唯一区别是, 算法 5 和算法 6 的总内存从 $O(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})$ 增加到 $\tilde{O}(n^{3-3(3-\omega)/(4-\alpha/2)-\alpha(\omega/2-1)})$. 查询从源点到任意其他顶点的距离后, 将所得距离乘以最小边权重, 即可得到从源点到该顶点的实际距离. 证毕.

5 扩展到动态 APSP 问题

所设计并行动态 SSSP 算法同样适用于求解完全动态 APSP 问题, 且无需 n 次重复执行算法 6. 通过对算法 6 进行修改和增加额外内存, 可以在相同的轮数复杂度内, 更新所有 n 个顶点之间距离. 接下来, 首先介绍两个已知的 MPC 算法[41].

引理 12. *MPC-MMOS*(A_1,A_2,B,n,α)[41]. 在 MPC 模型中, 每台机器内存大小为 $O(n^\alpha)$, 内存参数 $\alpha\in(0,1)$. 给定图 $G=(V,E)$, 其中 $|V|=n$, 以及两个 $n\times n$ 矩阵 A_1 和 A_2, 存在一个并行算法 *MPC-MMOS*(A_1,A_2,B,n,α), 可在 $O(\alpha^{-1})$ 轮内完成半环上矩阵乘法 $B=A_1\times A_2$. 该算法总内存需求为 $O(n^{3-\alpha/2})$.

MPC-MMOS(A_1,A_2,B,n,α) 算法流程与第 3.3 节中算法 2 相同, 只是去掉了 for 循环. 此外, 在 MPC 模型中计算半环上两个大小不同的矩阵乘法所需的轮数复杂度如下.

引理 13. *MPC-IMMOS*(A_1,A_2,B,n,n^y,α)[41]. 在每台机器内存大小为 $O(n^\alpha)$ 的 MPC 模型中, 内存参数 $\alpha\in(0,1)$. 给定图 $G=(V,E)$, $|V|=n$, 一个大小为 $n\times n^y$ 的矩阵 A_1 和一个大小为 $n^y\times n$ 的矩阵 A_2, $y\geqslant\alpha/2$. 存在一个并行算法 *MPC-IMMOS*(A_1,A_2,B,n,n^y,α), 可在 $O(\alpha^{-1})$ 轮内计算半环上两个大小不同的矩阵乘积 $B=A_1\times A_2$. 该算法的总内存需求为 $O(n^{2+(y-\alpha/2)})$.

更新 n 个顶点之间的距离则仅需修改算法 6 第 21、22 行, 具体如下.

- 第 21 行: 查询距离矩阵 D 以获取 n 个顶点到采样顶点集合 H 的 $\leqslant n^s$ 条边的最短路径长度, 并组合成一个大小为 $n\times(n^{1-s}\ln n)$ 的矩阵 M_1. 类似地, 查询距离矩阵 D 以获取采样顶点到 n 个顶点的 $\leqslant n^s$ 条边的最短路径长度, 并组合为大小为 $(n^{1-s}\ln n)\times n$ 的矩阵 M_2.
- 第 22 行: 将矩阵 M_1 分成 $n^s/\ln n$ 个子矩阵 M_1^i, 每个大小为 $(n^{1-s}\ln n)\times(n^{1-s}\ln n)$, 其中 $i\in\{1,2,\ldots,n^s/\ln n\}$. 然后对所有 $i\in\{1,2,\ldots,n^s/\ln n\}$, 执行子程序 *MPC-MMOS*$(M_1^i,H,MH^i,n^{1-s}\ln n,\alpha)$ (参考引理 12), 其中 MH^i 是一个 $(n^{1-s}\ln n)\times(n^{1-s}\ln n)$ 矩阵. 之后, 执行子程序 *MPC-IMMOS*$(MH,M_2,D_{\mathrm{APSP}},n,n^{1-s}\ln n,\alpha)$ (参考引理 13). 其中, MH 是一个 $n\times(n^{1-s}\ln n)$ 矩阵, 由 $n^s/\ln n$ 个大小为 $(n^{1-s}\ln n)\times(n^{1-s}\ln n)$ 的子矩阵 MH^i 组成; D_{APSP} 是一个 $n\times n$ 矩阵, 表示最终所有顶点之间距离.

接下来分析更新动态 APSP 问题中所有节点对之间距离所需的轮数复杂度和总内存.

定理 2. 给定一个无向/有向图 $G=(V,E)$, 其中 $|V|=n$, 边权重为整数且最大边权重和最小边权重的比值为 $\lceil poly(\log n)\rceil$. 更新操作为单条边的删除、插入或单条边权重修改. 在每台机器内存大小为 $O(n^\alpha)$ 且参数 $\alpha\in(0,1)$ 的 MPC 模型中, 存在一个并行算法, 以高概率在 $O(\log^3 n)$ 轮内预处理图 G, 并在 $O(\alpha^{-1}\log n)$ 轮内更新 n 个顶点之间距离. 查询任意两顶点之间距离需要 $O(1)$ 轮. 所需总内存为 $\tilde{O}(n^{2+(\omega-1)(1-\alpha/2)/2})$, 当 $\alpha\in(0,1)$ 且 $\omega\simeq 2.3728$ 时, 总内存介于 $\tilde{O}(n^{2.343})$ 和 $\tilde{O}(n^{2.686})$ 之间.

证明: 根据以上分析, 仅需计算并执行上述两步所需的轮数复杂度和总内存. 首先, 查询距离矩阵 D 以获得矩阵 M_1 和 M_2, 这一操作可在常数轮内完成. 子程序 *MPC-MMOS*$(M_1^i,H,MH^i,n^{1-s}\ln n,\alpha)$ 对所有 $i\in\{1,2,\ldots,n^s/\ln n\}$,

可并行运行, 根据引理 12, 其所需总内存为 $\tilde{O}(n^{(3-\alpha/2)(1-s)+s})$.至于子程序 $MPC\text{-}IMMOS(MH, M_2, D_{\text{APSP}}, n, n^{1-s}\ln n, \alpha)$, 根据引理 13, 其所需总内存为 $O(n^{3-s-\alpha/2})$. 结合引理 11, 更新所有顶点对之间距离的轮数复杂度为 $O(\alpha^{-1}\log n)$, 所需总内存为 $\tilde{O}(n^{3-s-\alpha/2})$.

最后, 将更新动态 APSP 距离所需总内存与预处理原始图所需总内存相结合 (引理 10). 当参数 $s=(3-\omega)(1/2-\alpha/4)$ 时, 满足引理 13 的条件 (当 $\alpha\in(0,1)$ 时, $1-s\geqslant\alpha/2$), 则更新动态 APSP 距离所需总内存为 $\tilde{O}(n^{2+(\omega-1)(1-\alpha/2)/2})$.

如第 4.3 节所述, 算法 6 第 1 部分用于更新所有顶点之间 $\leqslant n^s$ 条边的最短路径长度. 仅需修改算法 6 第 2 部分的第 21、22 行便可更新动态 APSP 问题中 $\geqslant n^s$ 条边的最短路径长度. 因此, 查询任意两顶点之间距离时, 需比较这两个顶点之间 $\leqslant n^s$ 条边的最短路径长度与 $\geqslant n^s$ 条边的最短路径长度. 然后, 取其中较小值, 作为最终结果, 此查询过程可在 $O(1)$ 轮内完成. 证毕.

值得一提的是, 由于预处理算法可同时用于动态 APSP 问题以及动态 SSSP 问题的图, 因此, 更新动态 APSP 所需总内存略高于更新动态 SSSP 问题所需总内存. 由于总内存由参数 s 的值决定 ($s\in(0,1)$), 动态 APSP 问题和动态 SSSP 问题的总内存差异较小.

此外, 我们针对动态 APSP 问题设计的并行算法与 Wang 等人[41]所采用的并行静态 APSP 问题具有相同轮数复杂度, 但总内存开销更少. 同时, 与 Karczmarz 等人[43]基于 PRAM 模型提出的深度为 $\tilde{O}(d)$ ($d\in[1,n]$) 且总内存为 $\tilde{O}(n^3)$ 的并行静态 APSP 算法相比, 所提并行动态 SSSP 算法在轮数复杂度和总内存上均更具优势.

6 结论与未来工作

本文针对 MPC 模型中完全动态精确单源最短路径问题, 设计了一种随机完全动态 MPC 算法, 其更新轮数复杂度为 $O(\alpha^{-1}\log n)$. 本文采用多种高效方法计算多项式矩阵的逆, 例如具有常数轮数复杂度的并行多项式乘积算法. 通过与现有 MPC 模型中静态近似或精确 SSSP 算法进行复杂度比较, 证明了所提并行更新 SSSP 算法求解动态精确 SSSP 问题的高效性. 此外, 通过对该算法稍作修改和扩展, 所提并行完全动态算法还能更新所有顶点对之间的距离. 这表明所提并行算法在动态单源最短路径长度之外具有潜在应用价值.

然而, 受限于 Sherman-Morrison 恒等式的秩-1 更新特性, 所提并行动态 SSSP 算法仅适用于单边更新场景. 若将其扩展至顶点更新操作 (即顶点及其关联边的插入或删除), 即多边更新, 则需多次调用该恒等式来更新源点到其他顶点的距离. 在此情形下, 随着更新边数增加 ($\omega(poly\log n)$), 距离更新所需的轮数复杂度将急剧增加, 无法保证对数多项式时间的更新效率.

迄今为止, 已证实代数方法能有效降低 MPC 模型中各类图问题的轮数复杂度, 但代价是增大内存需求. 尽管代数技术相对简单, 但仍有局限性, 例如在给定图中求解顶点之间最短路径的成本过高. 因此, 探索以低轮数复杂度高效求解最短路径问题的技术和方法具有重要意义.

References

[1] Dhulipala L, Durfee D, Kulkarni J, Peng R, Sawlani S, Sun XR. Parallel batch-dynamic graphs: Algorithms and lower bounds. In: Proc. of the 14th Annual ACM-SIAM Symp. on Discrete Algorithms. Salt Lake City: SIAM, 2020. 1300–1319. [doi: 10.1137/1.9781611975994.79]

[2] Nowicki K, Onak K. Dynamic graph algorithms with batch updates in the massively parallel computation model. In: Proc of the 2021 ACM-SIAM Symp. on Discrete Algorithms. SIAM, 2021. 2939–2958. [doi: 10.1137/1.9781611976465.175]

[3] Italiano GF, Lattanzi S, Mirrokni VS, Parotsidis N. Dynamic algorithms for the massively parallel computation model. In: Proc of the 31st ACM Symp. on Parallelism in Algorithms and Architectures. Phoenix: ACM, 2019. 49–58. [doi: 10.1145/3323165.3323202]

[4] Gilbert S, Lu LLE. How fast can you update your MST? In: Proc of the 32nd ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2020. 531–533. [doi: 10.1145/3350755.3400240]

[5] Ghaffari M, Nowicki K. Massively parallel algorithms for minimum cut. In: Proc of the 39th ACM Symp. on Principles of Distributed Computing. ACM, 2020. 119–128. [doi: 10.1145/3382734.3405737]

[6] Dinitz M, Nazari Y. Massively parallel approximate distance sketches. In: Proc of the 23rd Int'l Conf. on Principles of Distributed Systems. Neuchâtel: Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2019. 35: 1–35: 17. [doi: 10.4230/LIPICS.OPODIS.2019.35]

[7] Biswas AS, Dory M, Ghaffari M, Mitrović S, Nazari Y. Massively parallel algorithms for distance approximation and spanners. In: Proc of the 33rd ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2021. 118–128. [doi: 10.1145/3409964.3461784]

[8] Zhu XB, Li WJ, Yang YJ, Wang JX. Incremental algorithms for the maximum internal spanning tree problem. Science China Information Sciences, 2021, 64(5): 152103. [doi: 10.1007/S11432-019-2630-2]

[9] Sankowski P. Subquadratic algorithm for dynamic shortest distances. In: Proc of the 11th Annual Int'l Conf. on Computing and Combinatorics. Kunming: Springer, 2005. 461–470. [doi: 10.1007/11533719_47]

[10] Bodwin G, Krinninger S. Fully dynamic spanners with worst-case update time. In: Proc of the 24th Annual European Symp. on Algorithms. Aarhus: Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2016. 17: 1–17: 18. [doi: 10.4230/LIPICS.ESA.2016.17]

[11] Bernstein A, Gutenberg MP, Saranurak T. Deterministic decremental reachability, SCC, and shortest paths via directed expanders and congestion balancing. In: Proc of the 61st IEEE Annual Symp. on Foundations of Computer Science. Durham: IEEE, 2020. 1123–1134. [doi: 10.1109/FOCS46700.2020.00108]

[12] van den Brand J, Nanongkai D. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In: Proc of the 60th IEEE Annual Symp. on Foundations of Computer Science. Baltimore: IEEE, 2019. 436–455. [doi: 10.1109/FOCS.2019.00035]

[13] Henzinger M, Krinninger S, Nanongkai D. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In: Proc of the 55th IEEE Annual Symp. on Foundations of Computer Science. Philadelphia: IEEE, 2014. 146–155. [doi: 10.1109/FOCS.2014.24]

[14] Sankowski P. Dynamic transitive closure via dynamic matrix inverse [extended abstract]. In: Proc of the 45th Symp. on Foundations of Computer Science. Rome: IEEE, 2004. 509–517. [doi: 10.1109/FOCS.2004.25]

[15] Bernstein A, Forster S, Henzinger M. A deamortization approach for dynamic spanner and dynamic maximal matching. In: Proc. of the 30th Annual ACM-SIAM Symp. on Discrete Algorithms. San Diego: SIAM, 2019. 1899–1918. [doi: 10.1137/1.9781611975482.115]

[16] Li H, Liu YN, Yang SQ, Huang JB, Qiao SJ. Dynamic incremental graph partitioning algorithm based on vertex group redistribution. Ruan Jian Xue Bao/Journal of Software, 2024, 35(4): 1819–1840 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6842.htm [doi: 10.13328/j.cnki.jos.006842]

[17] Andoni A, Stein C, Zhong PL. Parallel approximate undirected shortest paths via low hop emulators. In: Proc. of the 52nd Annual ACM SIGACT Symp. on Theory of Computing. Chicago: ACM, 2020. 322–335. [doi: 10.1145/3357713.3384321]

[18] Forster S, Nanongkai D. A faster distributed single-source shortest paths algorithm. In: Proc of the 59th IEEE Annual Symp. Foundations of Computer Science. Paris: IEEE, 2018. 686–697. [doi: 10.1109/FOCS.2018.00071]

[19] Henzinger M, Krinninger S, Nanongkai D. Sublinear-time maintenance of breadth-first spanning trees in partially dynamic networks. ACM Trans. on Algorithms, 2017, 13(4): 51: 1–51: 24. [doi: 10.1145/3146550]

[20] Even S, Shiloach Y. An on-line edge-deletion problem. Journal of the ACM, 1981, 28(1): 1–4. [doi: 10.1145/322234.322235]

[21] Roditty L, Zwick U. On dynamic shortest paths problems. In: Proc of the 12th Annual European Symp. on Algorithms. Bergen: Springer, 2004. 580–591. [doi: 10.1007/978-3-540-30140-0_52]

[22] Elkin M, Matar S. Deterministic PRAM approximate shortest paths in polylogarithmic time and slightly super-linear work. In: Proc of the 33rd ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2021. 198–207. [doi: 10.1145/3409964.3461809]

[23] Cao NR, Fineman JT. Parallel exact shortest paths in almost linear work and square root depth. In: Proc. of the 2023 Annual ACM-SIAM Symp. on Discrete Algorithms. Florence: SIAM, 2023. 4354–4372. [doi: 10.1137/1.9781611977554.CH166]

[24] van den Brand J, Karczmarz A. Deterministic fully dynamic SSSP and more. In: Proc of the 64th IEEE Annual Symp. on Foundations of Computer Science. Santa Cruz: IEEE, 2023. 2312–2321. [doi: 10.1109/FOCS57990.2023.00142]

[25] Henzinger M, Krinninger S, Nanongkai D. A subquadratic-time algorithm for decremental single-source shortest paths. In: Proc. of the 25th Annual ACM-SIAM Symp. on Discrete Algorithms. Portland: SIAM, 2014. 1053–1072. [doi: 10.1137/1.9781611973402.79]

[26] Bernstein A, Chechik S. Deterministic decremental single source shortest paths: Beyond the O(mn) bound. In: Proc. of the 48th Annual ACM Symp. on Theory of Computing. Cambridge: ACM, 2016. 389–397. [doi: 10.1145/2897518.2897521]

[27] Bernstein A. Deterministic partially dynamic single source shortest paths in weighted graphs. In: Proc of the 44th Int'l Colloquium on Automata, Languages, and Programming. Warsaw: Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2017. 44: 1–44: 14. [doi: 10.4230/LIPICS.ICALP.2017.44]

[28] Bernstein A, Chechik S. Deterministic partially dynamic single source shortest paths for sparse graphs. In: Proc. of the 28th Annual ACM-SIAM Symp. on Discrete Algorithms. Barcelona: SIAM, 2017. 453–469. [doi: 10.1137/1.9781611974782.29]

[29] Chuzhoy J, Saranurak T. Deterministic algorithms for decremental shortest paths via layered core decomposition. In: Proc. of the 2021 ACM-SIAM Symp. on Discrete Algorithms. SIAM, 2021. 2478–2496. [doi: 10.1137/1.9781611976465.147]

[30] Chuzhoy J, Khanna S. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut

problems. In: Proc. of the 51st Annual ACM SIGACT Symp. on Theory of Computing. Phoenix: ACM, 2019. 389–400. [doi: 10.1145/3313276.3316320]

[31] Chechik S, Zhang TY. Incremental single source shortest paths in sparse digraphs. In: Proc. of the 2021 ACM-SIAM Symp. on Discrete Algorithms. SIAM, 2021. 2463–2477. [doi: 10.1137/1.9781611976465.146]

[32] Ullman JD, Yannakakis M. High-probability parallel transitive-closure algorithms. SIAM Journal on Computing, 1991, 20(1): 100–125. [doi: 10.1137/0220006]

[33] Rozhoň V, Grunau C, Haeupler B, Zuzic G, Li J. Undirected (1+ε)-shortest paths via minor-aggregates: Near-optimal deterministic parallel and distributed algorithms. In: Proc. of the 54th Annual ACM SIGACT Symp. on Theory of Computing. Rome: ACM, 2022. 478–487. [doi: 10.1145/3519935.3520074]

[34] Goodrich MT, Sitchinava N, Zhang Q. Sorting, searching, and simulation in the mapreduce framework. In: Proc. of the 22nd Int'l Symp. on Algorithms and Computation. Yokohama: Springer, 2011. 374–383. [doi: 10.1007/978-3-642-25591-5_39]

[35] Ingole A, Nasre R. Dynamic shortest paths using JavaScript on GPUs. In: Proc of the 22nd IEEE Int'l Conf. High-performance Computing. Bengaluru: IEEE, 2015. 1–5.

[36] Khanda A, Srinivasan S, Bhowmick S, Norris B, Das SK. A parallel algorithm template for updating single-source shortest paths in large-scale dynamic networks. IEEE Trans. on Parallel and Distributed Systems, 2022, 33(4): 929–940. [doi: 10.1109/TPDS.2021.3084096]

[37] Karloff HJ, Suri S, Vassilvitskii S. A model of computation for MapReduce. In: Proc. of the 21st Annual ACM-SIAM Symp. on Discrete Algorithms. Austin: SIAM, 2010. 938–948. [doi: 10.1137/1.9781611973075.76]

[38] Le Gall F. Powers of tensors and fast matrix multiplication. In: Proc. of the 39th Int'l Symp. on Symbolic and Algebraic Computation. Kobe: ACM, 2014. 296–303. [doi: 10.1145/2608628.2608664]

[39] van den Brand J, Saranurak T. Sensitive distance and reachability oracles for large batch updates. In: Proc. of the 60th IEEE Annual Symp. on Foundations of Computer Science. Baltimore: IEEE, 2019. 424–435. [doi: 10.1109/FOCS.2019.00034]

[40] Deng MH, Ramanan P. MapReduce implementation of Strassen's algorithm for matrix multiplication. In: Proc. of the 4th Annual ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond. Chicago: ACM, 2017. 7: 1--7: 10. [doi: 10.1145/3070607.3070614]

[41] Wang CL, Hua QS, Jin H, Zheng CD. Massively parallel algorithms for fully dynamic all-pairs shortest paths. Frontiers of Computer Science, 2024, 18(4): 184611. [doi: 10.1007/S11704-024-3452-2]

[42] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. 3rd ed., Cambridge: The MIT Press, 2009.

[43] Karczmarz A, Sankowski P. A deterministic parallel APSP algorithm and its applications. In: Proc. of the 2021 ACM-SIAM Symp. on Discrete Algorithms. SIAM, 2021. 255–272. [doi: 10.1137/1.9781611976465.17]

附中文参考文献

[16] 李贺, 刘延娜, 杨舒琪, 黄健斌, 乔少杰. 基于顶点组重分配的动态增量图划分算法. 软件学报, 2024, 35(4): 1819–1840. http://www.jos.org.cn/1000-9825/6842.htm [doi: 10.13328/j.cnki.jos.006842]

作者简介

王迟蕾, 博士生, CCF 学生会员, 主要研究领域为动态图算法, 分布式并行计算.

华强胜, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为并行分布式计算理论, 算法与系统.

金海, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为高性能计算, 网格计算, 云计算和虚拟化.