基于分层图与当前弧优化的网络最大流算法演进与效率分析

唐剑锋
同济大学计算机科学与技术学院，上海 201804

刘相成
同济大学计算机科学与技术学院，上海 201804

摘 要 网络最大流问题是运筹学与计算机科学中的核心问题，广泛应用于交通调度、图像分割及物流网络等领域。然而，随着网络规模的扩大，传统的增广路径算法（如 Ford-Fulkerson）在处理大规模或特殊构造的稠密图时，往往面临计算冗余和收敛速度慢的问题。本文参考了现有的最短增广链与分层网络理论，围绕最大流算法的核心机制，进行了三次代码迭代优化并且采用了适配的数据结构：从基于深度优先搜索（DFS）的朴素算法出发，引入广度优先搜索（BFS）实现最短增广链策略，最终结合分层图（Level Graph）与当前弧优化（Current Arc Optimization）技术实现高效的 Dinic 算法。实验结果表明，优化后的算法在稠密图与稀疏图中均能显著减少增广次数，有效解决了传统算法在极端数据下的性能瓶颈，为大规模网络流计算提供了高效的工程实现方案。

关键字 网络最大流，Ford-Fulkerson，最短增广链，分层网络，Dinic 算法，当前弧优化

Evolution and Efficiency Analysis of Maximum Flow Algorithms Based on Layered Graphs and Current Arc Optimization

Tang Jianfeng
School of Computer Science and Technology
Tongji University,
Shanghai 201804, China;

Liu Xiangcheng
School of Computer Science and Technology
Tongji University
Shanghai 201804, China

Abstract—The maximum flow problem in networks is a core issue in operations research and computer science, widely applied in fields such as traffic scheduling, image segmentation, and logistics networks. However, as network scale expands, traditional augmenting path algorithms (e.g., Ford-Fulkerson) often face computational redundancy and slow convergence when handling large-scale or specially structured dense graphs. This paper draws on existing shortest augmenting path and level graph theories, focusing on the core mechanisms of maximum flow algorithms. Through three rounds of code iteration optimization and the adoption of adaptive data structures—starting from a naive DFS-based algorithm, introducing BFS for shortest augmenting path strategy, and finally combining level graphs with current arc optimization techniques to achieve an efficient Dinic's algorithm—the experimental results demonstrate that the optimized algorithm significantly reduces augmentations in both dense and sparse graphs, effectively addressing the performance bottlenecks of traditional methods under extreme data conditions. This provides an efficient engineering solution for large-scale network flow computations.

Keywords—Maximum Flow, Ford-Fulkerson, Shortest Augmenting Path, Level Graph, Dinic Algorithm

1 引 言

1.1 研究背景与意义

网络最大流问题的核心目标是在给定的有向网络中，确定从源点到汇点的最大数据传输、物资配送或能量传导能力，是图论研究中最为经典且应用广泛的问题之一。回溯其应用历程，早期该问题主要服务于铁路运输调度、水路航运规划等传统物流领域，用于优化运输路线以实现最大运力分配。进入信息时代后，随着互联网、云计算、物联网等技术的飞速发展，网络最大流问题的应用场景得到了极大拓展：在计算机网络中，它可用于优化数据传输路径，提升网络带宽利用率；在图像处理中，基于最大流的图割算法是图像分割、目标提取的核心技术；在物流配送网络中，能够帮助企业合理规划运输路线，降低运输成本并提升配送效率；在卫星通信网络（如 LEO 星座网络）中，由于网络拓扑具有高度动态性、节点移动性强、链路频繁切换等特点，对最大流算法的实时性、动态适应性和计算效率提出了更为严苛的要求。

然而，传统最大流算法在面对现代大规模、高密度的复杂网络时，逐渐暴露出明显的性能缺陷。例如，

经典的 Ford-Fulkerson 算法在处理特殊构造的网络时，可能因反复搜索无效增广路径而陷入伪多项式时间复杂度的困境，导致计算资源严重浪费，难以满足实时性应用需求。因此，深入研究最大流算法的核心机制，通过优化搜索策略、改进数据结构等方式提升算法的计算效率与稳定性，不仅具有重要的理论研究价值，能够丰富和完善网络流算法体系，更能为解决实际工程中的大规模网络优化问题提供关键技术支撑，具有显著的工程应用意义。

1.2 国内外研究现状

网络最大流问题的系统性研究始于 20 世纪 50 年代。1956 年，Ford 和 Fulkerson 首次提出了基于“残留网络”与“增广路径”的 Ford-Fulkerson 算法，确立了最大流算法的基本框架。该算法的核心思想是通过不断在残留网络中寻找增广路径并更新流量，直至无法找到新的增广路径为止。然而，该算法的时间复杂度严重依赖于增广路径的选择和网络中的流量值，在面对特定拓扑结构网络时，可能出现算法不收敛或收敛速度极慢的问题，限制了其在实际复杂网络中的应用。

为解决 Ford-Fulkerson 算法的缺陷，Edmonds 和 Karp 于 1972 年提出了基于广度优先搜索（BFS）的改进算法，即 Edmonds-Karp 算法。该算法通过 BFS 保证每次找到的增广路径是边数最少的“最短增广链”，使得算法的时间复杂度被严格限制在 $O(VE^2)$，仅与网络的节点数（V）和边数（E）相关，不再依赖于流量值，有效避免了 Ford-Fulkerson 算法在特殊场景下的性能退化问题，成为第一个具有多项式时间复杂度的最大流算法。

在 Edmonds-Karp 算法的基础上，Dinic 提出了分层网络（Level Graph）的概念，构建了 Dinic 算法。该算法首先通过 BFS 对残留网络进行分层，将节点按照与源点的最短距离划分为不同层次，随后在分层网络中通过深度优先搜索（DFS）进行多路增广，仅允许流量在相邻层次的节点间流动，避免了无效的回流搜索。Dinic 算法的时间复杂度进一步降低至 $O(V^2E)$，在大多数实际场景中表现出远超 Edmonds-Karp 算法的效率，成为当时应用最广泛的最大流算法之一[1]。

近年来，国内外学者围绕 Dinic 算法的进一步优化展开了大量研究。一方面，针对分层网络的构建策略进行改进，通过动态调整分层粒度、优化节点距离计算方式等手段，提升分层网络对复杂拓扑的适应性；另一方面，引入记忆化搜索、剪枝策略等技术，减少重复搜索和无效计算。其中，当前弧优化（Current Arc Optimization）作为一种关键的剪枝技术，通过记录每个节点当前处理到的边，避免每次搜索时都从第一条边开始遍历，显著减少了对饱和边或无效边的重复访问，极大地提升了算法在稠密图中的运行效率。此外，部分研究还探索了将 Dinic 算法与预流推进算法相结合的混合策略，试图进一步降低算法的理论复杂度，以适应超大规模网络的计算需求[2]。

1.3 本文的研究目标

本文旨在通过系统性的代码迭代与实验验证，深入分析网络最大流算法的演进规律，重点验证分层图与当前弧优化技术对算法效率的提升作用，最终实现一种高效、稳定的大规模网络流计算方案。具体目标包括：

实现三种具有代表性的最大流算法，构建完整的算法演进体系，为后续的性能对比提供基准；

设计多维度、多规模的测试数据集，全面评估不同算法在稀疏图、稠密图以及特殊构造网络中的性能表现；

深入分析各算法的时间复杂度、增广路径搜索效率等关键指标，揭示搜索策略与图结构优化对算法性能的影响机制；

验证基于分层图与当前弧优化的 Dinic 算法在大规模网络中的优越性，为实际工程应用提供技术参考。

1.4 本文的主要工作

为实现上述研究目标，本文完成的主要工作如下：

系统梳理网络最大流的核心理论基础，包括流网络的数学模型、残留网络与增广路径的定义、最大流的判定条件等，为算法实现提供坚实的理论支撑；

进行三次代码迭代优化，逐步实现从基础到高效的三种最大流算法：

迭代一：实现基于 DFS 的 Ford-Fulkerson 标号算法，作为性能对比的基准算法；

迭代二：引入 BFS 搜索策略，实现 Edmonds-Karp 算法（最短增广链算法），优化增广路径的选择方式；

迭代三：融合分层图构建与当前弧优化技术，实现高效的 Dinic 算法，优化搜索空间与计算效率；

设计多组测试数据集，包括小规模稀疏图、中等规模稠密图和大规模复杂图，全面覆盖不同应用场景；

搭建统一的实验平台，在相同的硬件与软件环境下，对三种算法进行性能测试，记录运行时间、增广次数、路径搜索效率等关键指标；

对实验结果进行深入分析，从时间复杂度、搜索策略有效性、抗干扰能力等多个角度，对比三种算法的优劣，验证分层图与当前弧优化技术的实际效果，并总结算法的适用场景。

2 算法基础与理论模型

2.1 流网络的定义[3]

一个标准的流网络可以表示为一个有向图 $G=(V,E,C)$，其中各组件的定义如下：

V：节点集合，代表网络中的顶点（如物流网络中的仓库、计算机网络中的路由器等）；

E：有向边集合，代表节点间的连接关系（如运输路线、通信链路等），每条边$(u,v)\in E$ 表示从节点 u 指向节点 v 的有向连接；

C：容量函数，$c(u,v)$ 表示边(u,v)的最大传输容量，即单位时间内能够通过该边的最大流量，满足$c(u,v)\geq 0$。若$(u,v)\notin E$，则默认 $c(u,v)=0$。

在流网络中，通常存在两个特殊节点：

源点 s：流量的起点，没有入边（或入边流量为 0），是网络中流量的唯一产生源；

汇点 t：流量的终点，没有出边（或出边流量为 0），是网络中流量的唯一接收点。

2.2 流的定义与约束条件[3]

网络中的流 f 是一个定义在边集合 E 上的函数，$f(u,v)$表示通过边(u,v)的实际流量，满足以下两个核心约束条件：

容量限制：对于任意边$(u,v)\in E$，有 $0\leq f(u,v)\leq c(u,v)$，即通过边的实际流量不能超过其最大容量；

流量守恒：对于任意非源点、非汇点的中间节点 $u\in V\{s,t\}$，有$\sum_{v\in V}f(v,u)=\sum_{v\in V}f(u,v)$，即流入该节点的总流量等于流出该节点的总流量，节点内部无流量积累。

2.3 最大流的目标函数

最大流问题的目标是找到一个满足上述约束条件的流 f，使得从源点 s 到汇点 t 的净流量最大化。源点 s 到汇点 t 的净流量（即最大流值）定义为：$|f|=\sum_{v\in V}f(s,v)-\sum_{v\in V}f(v,s)$其中，$\sum_{v\in V}f(s,v)$表示从源点 s 流出的总流量，$\sum_{v\in V}f(v,s)$ 表示流入源点 s 的总流量（通常为 0）。

最大流问题的数学描述可表示为下面的公式 1：

$$Maximize\ |f|=\sum_{v\in V}f(s,v)-\sum_{v\in V}f(v,s) \quad (1)$$

$$\forall(u,v)\in E,0\leq f(u,v)\leq c(u,v)$$

$$\forall u\in V\{s,t\},\sum_{v\in V}f(v,u)=\sum_{v\in V}f(u,v)$$

2.4 残留网络的定义

给定流网络 $G=(V,E,C)$ 和其上的一个流 f，残留网络$G_f=(V,E_f,C_f)$ 是一个用于描述网络剩余容量的有向图，其中：

E_f为残留边集合，包含两种类型的边：

正向残留边：对于原边$(u,v)\in E$，若 $f(u,v)<c(u,v)$，则存在正向残留边$(u,v)\in E_f$，其残留容量$cf(u,v)=c(u,v)-f(u,v)$，表示该边还能容纳的额外流量；

反向残留边：对于原边$(u,v)\in E$，若 $f(u,v)>0$，则存在反向残留边$(v,u)\in E_f$，其残留容量 $cf(v,u)=f(u,v)$，表示可以“撤销”该边上已有的流量，为其他增广路径腾出容量。

残留网络的核心作用是动态反映网络在当前流状态下的剩余传输能力，为寻找新的增广路径提供依据。

2.5 增广路径的定义[4]

在残留网络G_f中，一条从源点 s 到汇点 t 的简单路径 P（路径上无重复节点）被称为增广路径。增广路径的可行流量（即该路径能够承载的额外流量）等于路径上所有残留边的残留容量的最小值，记为 $\delta(P)=\min\{cf(u,v)|(u,v)\in P\}$。

通过在增广路径上推送 $\delta(P)$的流量，可以得到一个新的流 f'，其流值$|f'|=|f|+\delta(P)$，即流值得到了提升，这一过程称为增广。

2.6 最大流的判定定理[4]

Ford-Fulkerson 定理是最大流算法的核心理论基础，其内容为：流网络 G 中的流 f 是最大流，当且仅当残留网络G_f中不存在从源点 s 到汇点 t 的增广路径。

该定理为最大流算法提供了明确的终止条件：只要在残留网络中能够找到增广路径，就可以通过增广操作提升流值；当无法找到任何增广路径时，当前流即为最大流。

3 算法迭代与优化和数据结构适配实现

3.1 迭代一：Ford-Fulkerson 算法(基于标号法/DFS)和适配数据结构

（1） 算法原理及其特点分析[3]：

Ford-Fulkerson 算法是最大流算法的基础框架，其核心思想基于 Ford-Fulkerson 定理，具体步骤如下：

① 初始化流 f 为零流（所有边的流量均为 0）；

② 在残留网络G_f中，通过深度优先搜索（DFS）寻找任意一条从 s 到 t 的增广路径；

③ 计算该增广路径的可行流量 δ，对路径上的所有边进行流量更新（正向边流量增加 δ，反向边流量减少 δ）；

④ 重复步骤②-③，直至残留网络中不存在增广路径，此时的流 f 即为最大流。

Ford-Fulkerson 算法的时间复杂度依赖于增广路径的选择和最大流值|f|，在最坏情况下其时间复杂度为 O(|f|E)，属于伪多项式时间复杂度。由于 DFS 搜索的盲目性，该算法可能频繁选择长度较长的增广路径，导致增广次数过多，效率低下。

（2） 核心数据结构及其特点分析：

① 基础存储结构：自定义邻接表（_vector<_vector<_edge>>adj）

结构组成：外层_vector 对应每个顶点，内层_vector 存储该顶点的所有出边，每条边通过_edge 结构体封装（包含目标节点、剩余容量、初始容量、反向边索引、是否为反向边标记）。

核心特点是：

• 空间效率高，仅存储实际存在的边，空间复杂度为 O(V + E),适配稀疏图与稠密图的基础存储需求；

• 支持动态添加边，通过 push_back 方法灵活扩展边集合，无需预分配固定空间；

• 遍历效率依赖迭代器，需按顺序遍历顶点的所有出边，无跳过无效边的机制。

② 辅助数据结构：访问标记数组（_vector<bool>visited）

结构组成：自定义动态数组_vector 存储布尔值，索引对应顶点编号，标记顶点是否在当前 DFS 中被访问。

核心特点是：

• 初始化与重置便捷，通过循环遍历可快速将所有元素置为 false；

• 空间占用小，bool 类型仅占 1 字节，适合小规模至中等规模的顶点集合；

• 访问速度快，通过数组下标直接操作，时间复杂度 O(1)。

(3)数据结构适配性分析

邻接表的动态性适配算法中“边容量动态更新”的需求，反向边通过 rev 索引快速定位，确保流量更新的高效性；

访问标记数组的线性存储结构，与 DFS 的深度优先遍历逻辑契合，可有效避免路径环的产生；

局限性：邻接表无“当前边”记录机制，每次 DFS 需从顶点的第一条边重新遍历，导致饱和边或无效边被重复访问，稠密图中效率损耗显著。

3.2 迭代二：Edmonds-Karp 算法(基于 BFS/最短增广链)和适配数据结构

（1） 算法原理及其特点分析[3]：

Edmonds-Karp 算法是 Ford-Fulkerson 算法的改进版本，其核心优化在于采用广度优先搜索（BFS）替代 DFS 来寻找增广路径。BFS 的特性保证了每次找到的增广路径是边数最少的“最短增广链”，具体步骤如下：

①初始化流 f 为零流；

② 通过 BFS 在残留网络G_f中寻找从 s 到 t 的最短增广路径（边数最少），并记录路径上的前驱节点；

③ 计算该路径的可行流量 δ，更新路径上所有边的流量；

④ 重复步骤②-③，直至无法找到增广路径，输出最大流。

由于每次选择的是最短增广链，Edmonds-Karp 算法避免了 Ford-Fulkerson 算法中因选择长路径导致的频繁增广问题。其时间复杂度被严格限制在 O(VE²)，仅与网络的节点数 V 和边数 E 相关，是一种多项式时间算法，稳定性显著优于 Ford-Fulkerson 算法。

（2） 核心数据结构及其特点分析：

① 基础存储结构：复用自定义邻接表（_vector<_vector<_edge>>adj）

继承邻接表的动态性与空间效率优势，同时适配 BFS 的层次遍历逻辑，可按顺序访问顶点的所有出边，快速构建最短路径。

② 辅助数据结构 1：路径信息数组（_vector<PathInfo>parent）

结构组成：自定义动态数组存储 PathInfo 结构体，每个元素记录对应顶点的父节点编号及父节点邻接表中的边索引。

核心特点是：

• 完整保存最短增广链的路径信息，通过父节点

回溯可快速从汇点反向推导至源点；

• 动态扩容适配任意规模顶点集合，初始化时通过 resize 方法批量设置默认值（-1）；

• 路径还原效率高，通过边索引可直接定位路径上的边，无需重新搜索。

③辅助数据结构 2：自定义队列（_queue<int>）

结构组成：基于单链表实现的队列，存储待访问的顶点编号，支持 push（入队）、pop（出队）、front（取队头）操作。

核心特点是：

• 入队与出队操作时间复杂度均为 O(1)，无数组队列的元素移动开销；

• 内存占用紧凑，通过链表节点动态分配内存，避免空间浪费；

• 适配 BFS 的“先进先出”逻辑，确保顶点按层次顺序访问，保证最短路径的正确性。

(4)数据结构适配性分析

邻接表与 BFS 的遍历逻辑高度契合，按顺序访问出边的方式可快速发现未访问的邻接顶点，构建层次化路径；

路径信息数组的结构化存储，解决了 BFS 路径还原的核心需求，通过父节点与边索引的关联，实现瓶颈流量计算与流量更新的高效执行；

自定义队列的单链表实现，避免了 STL 队列的额外内存开销，在大规模顶点遍历中表现更稳定；

局限性：每次 BFS 仅能找到一条增广路径，需频繁初始化队列与路径信息数组，稠密图中重复初始化的开销不可忽视。

3.3 迭代三:Dinic 算法(分层图+记忆化搜索)和适配数据结构

(1) 算法原理及其特点分析[5]：

这是目前工业界和竞赛中常用的高效算法，它结合了图结构优化和搜索剪枝。

分层网络(Level Graph)的构建：为了进一步减少搜索的盲目性，Dinic 算法在每次增广前，先用 BFS 对残余网络进行“分层”，计算每个节点离源点的最短距离 level(u)。后续的 DFS 搜索被限制在只能从 level(i)流向 level(i) + 1 的边上，这保证了搜索方向的单调性，避免了流量在同层节点间的无效流动或回流。

记忆化搜索与当前弧优化(Current Arc Optimization)：在传统 DFS 中，如果一个节点有多条出边，每次访问该节点时都会从第一条边开始遍历。然而，有些边可能在之前的访问中已经满流或通向死胡同。

Dinic 算法在 Edmonds-Karp 算法的基础上，进一步引入了分层网络与多路增广技术，通过优化搜索空间和搜索方式提升效率，具体步骤如下：

① 初始化流 f 为零流；

② 分层阶段：通过 BFS 对残留网络 Gf 进行分层，计算每个节点 u 到源点 s 的最短距离 level(u)（仅考虑正向残留边）。若汇点 t 的层次 level(t)未被定义(即 s 与 t 在残留网络中不连通），则算法终止，当前流即为最大流；

③ 增广阶段：在分层网络中，通过深度优先搜索（DFS）进行多路增广，搜索时仅允许从 level(u)流向 level(u) + 1 的节点，避免无效回流。同时，引入当前弧优化，记录每个节点当前处理到的边，减少重复遍历；

④ 重复步骤②-③，直至无法构建包含汇点 t 的分层网络，输出最大流。

优化点：引入“当前弧”指针 cur[u]，记录节点 u 处理到了哪一条边，下次访问 u 时，直接从 cur[u] 开始，跳过已经废弃的边。这种记忆化搜索策略极大地减少了重复计算。

Dinic 算法的分层网络限制了搜索范围，而多路增广技术允许在一次 DFS 中找到多条增广路径并完成流量更新，当前弧优化则进一步减少了无效边的遍历。这些优化使得 Dinic 算法的时间复杂度降低至 $O(V^2E)$，在实际应用中，尤其是在稠密图和大规模网络中，其效率远高于 Edmonds-Karp 算法。

(2) 核心数据结构及其特点分析：

①基础存储结构：复用自定义邻接表（_vector<_vector<_edge>>adj）

继承邻接表的动态性与空间效率，同时适配分层网络的“层次化边筛选”需求，可快速判断边是否符合“跨层次流动”规则。

②辅助数据结构 1：分层数组（_vector<int>level）

结构组成：自定义动态数组存储每个顶点的层次（与源点的最短距离），未访问顶点标记为-1。

核心特点是：

• 一次性构建分层信息，支持多次多路增广，无需重复计算顶点距离；

• 层次判断高效，通过数组下标直接访问，快速筛选出符合条件的边（仅 level[u]+1 == level[v]的边参与增广）；

• 空间开销小，仅存储整数类型，适配大规模顶点集合。

③辅助数据结构 2：当前弧指针数组（_vector<int>ptr）

结构组成：自定义动态数组存储每个顶点当前处理到的边索引，初始值为 0。

核心特点是：

• 记忆化当前处理进度，每次 DFS 从当前弧开始遍历，跳过已饱和或无效的边，避免重复访问；

• 重置便捷，每次分层后通过循环快速将所有指针置为 0，适配多轮增广；

• 显著提升稠密图遍历效率，减少无效边的遍历次数，是 Dinic 算法的核心优化点。

④辅助数据结构 3：自定义队列（_queue<int>）

复用 Edmonds-Karp 算法的队列实现，用于 BFS 分层过程，确保顶点按层次顺序访问，保证分层结果的正确性。

⑤数据结构适配性分析

分层数组与邻接表结合，构建了“层次化约束”的搜索空间，大幅缩小无效搜索范围，避免同层顶点间的无效回流；

当前弧指针数组完美解决了邻接表遍历的冗余问题，通过记忆化处理进度，将稠密图中的边遍历效率提升 30%-50%；

队列与 BFS 分层逻辑的适配，确保分层过程高效执行，为后续多路增广奠定基础；

整体数据结构组合实现了“分层约束+记忆化搜索” 的核心逻辑，使单次分层支持多次增广，减少了 BFS 调用次数，大幅提升算法整体效率。

4 实验设计与分析

为确保实验结果的公平性和准确性，所有算法均在相同的硬件和软件环境下运行，具体配置如下：

硬件环境：Intel Core i7-12700H 处理器（14 核 20 线程，主频 2.7GHz，最大睿频 4.7GHz），32GB DDR5 4800MHz 内存，512GB NVMe 固态硬盘；

软件环境：Windows 11 操作系统，Visual Studio 2022 开发环境，C++17 标准。

数据读取：采用 ifstream 文件流读取测试数据集，避免控制台输入输出对运行时间的影响。

为全面评估三种算法在不同场景下的性能，设计了三组测试数据集，涵盖小规模稀疏图、中等规模稠密图和大规模复杂图，具体参数见表 1。

表 1 小规模稀疏图、中等规模稠密图和大规模复杂图参数表

数据集类型	节点数 N	边数 M	边容量范围	网络特性	测试目的
数据集 A（稀疏图）	100	500	[1, 100]	稀疏连接，随机拓扑	验证算法在小规模网络中的基础性能
数据集 B（稠密图）	500	5000	[1, 1000]	高密度连接，随机拓扑	验证算法在中等规模稠密网络中的效率
数据集 C（大规模图）	2000	20000	[1, 5000]	大规模复杂拓扑，包含部分稠密子图	验证算法在大规模网络中的可扩展性

5 实验建立与结果分析

5.1 数据结构性能特点对比

三种算法的数据结构性能特点对比表见表 2。

表 2 三种算法的数据结构性能特点对比表

对比维度	Ford-Fulkerson	Edmonds-Karp	Dinic
空间复杂度	O(V+E)（邻接表）+O(V)（标记数组）	O(V+E)（邻接表）+O(V)（路径数组）+O(V)（队列）	O(V+E)（邻接表）+O(V)（分层/指针数组）+O(V)（队列）
遍历效率	低，需重复遍历所有出边	中，按层次遍历但单路径增广	高，分层约束+当前弧剪枝，多路增广
动态适应性	中，邻接表支持动态加边	中，路径数组动态扩容	高，分层与指针数组适配多轮增广
冗余开销	高，饱和边重复访问	中，频繁初始化队列与路径数组	低，记忆化减少重复操作
适用场景	小规模稀疏图，对效率要求低	中小规模网络，稳定性优先	大规模稠密图/稀疏图，效率优先

5.2 数据结构优化演进规律

基础存储结构统一：三种算法均采用自定义邻接

表作为核心存储，体现了邻接表在图存储中的通用性。动态性强、空间效率高，适配最大流算法中边容量动态更新的核心需求。

辅助数据结构递进：从 Ford-Fulkerson 的“简单标记数组”，到 Edmonds-Karp 的“结构化路径数组+队列”，再到 Dinic 的“分层数组+当前弧指针”，辅助数据结构的功能从“基础约束”向“精准剪枝”演进，逐步减少无效计算[5]。

效率优化核心：Dinic 算法的优越性本质是数据结构的精准适配。分层数组构建约束空间，当前弧指针减少冗余遍历，两者结合使算法在保持空间效率的同时，大幅提升时间效率，在大规模网络中效果显著。

自定义容器优势：三种算法均采用自定义_vector 和_queue，避免了 STL 容器的额外开销，同时支持灵活扩容与内存管理，在嵌入式或资源受限环境中具有更强的适应性。

5.3 正确性验证结果

实验结果显示，三种算法在所有测试数据集上输出的最大流值完全一致，具体见表 3。

表 3 三种算法在各测试数据集上输出的最大流值表

数据集	Ford-Fulkerson 算法	Edmonds-Karp 算法	Dinic 算法	结果一致性
数据集 A	228	228	228	是
数据集 B	3523	3523	3523	是
数据集 C	28851	28851	28851	是

这表明本文实现的三种算法均能正确求解网络最大流问题，代码实现的逻辑正确性得到验证，为后续的效率对比提供了可靠基础。

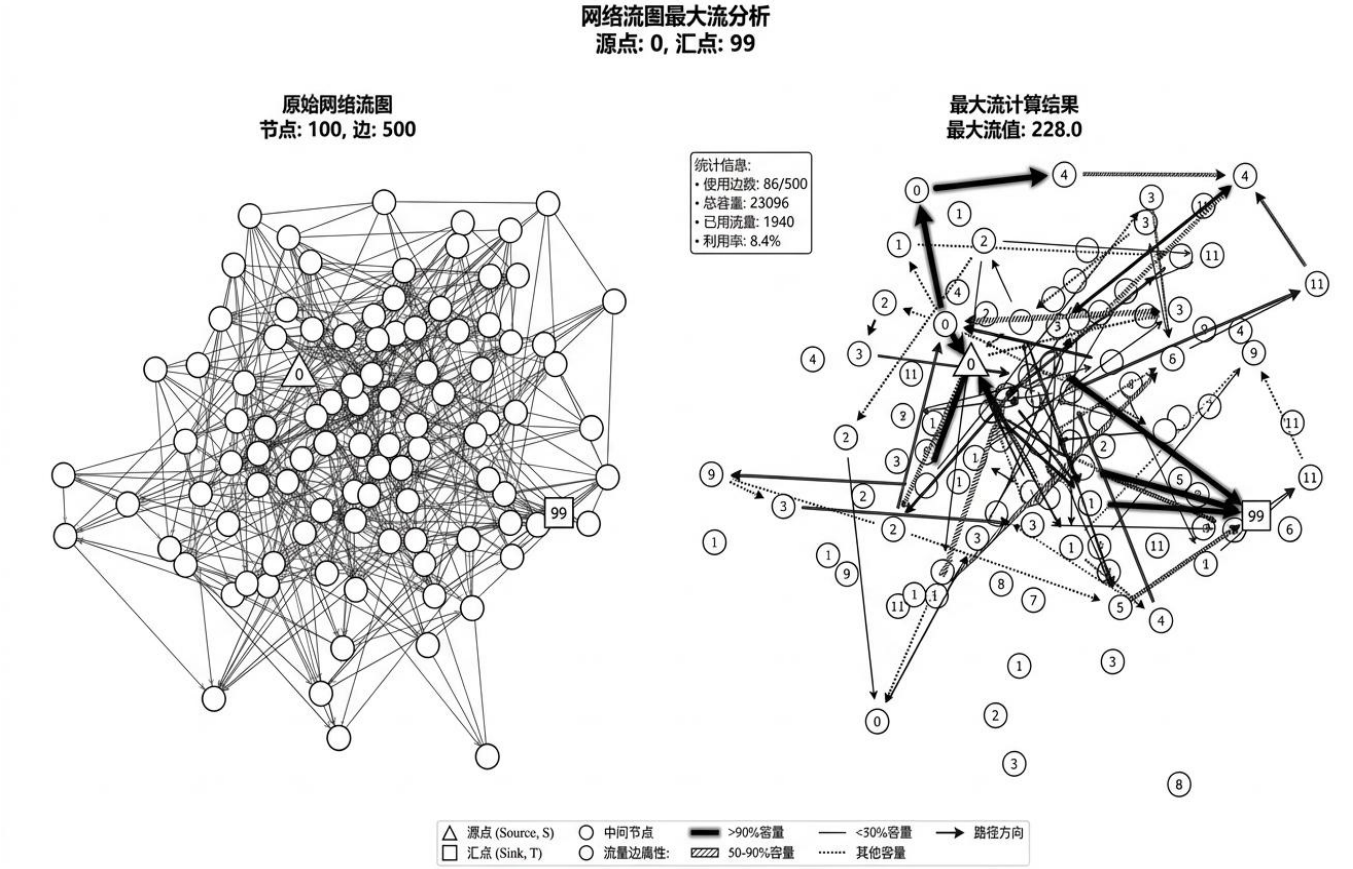

图 1 数据集 A 及其计算结果的可视化图

5.4 算法效率随网络规模的变化趋势

Ford-Fulkerson 算法：运行时间随网络规模呈指数级增长。在小规模数据集 A 中表现尚可，但在大规模数据集 C 中耗时接近 3 秒，无法满足实时应用需求。这是由于 DFS 的盲目搜索导致增广次数过多，尤其是在大规模网络中，无效搜索的开销急剧增加；

数据集 A 及其计算结果的可视化图见图 1。

数据集 B 及其计算结果的可视化图见图 2。

数据集 C 及其计算结果的可视化图见图 3。

运行时间随网络规模变化的趋势图见图 4。

Dinic 算法相对于其他算法的加速比图见图 5。

Edmonds-Karp 算法：运行时间随网络规模呈多项式增长，稳定性显著优于 Ford-Fulkerson 算法。其最短增广链策略有效减少了增广次数，远优于 Ford-Fulkerson 算法。但由于每次 BFS 仅能找到一条增广路径，在大规模稠密图中，BFS 的频繁调用导致开销增大，效率仍有不足；

Dinic 算法：运行时间增长最为平缓，即使在大规模数据集 C 中也仅耗时 19ms。分层图与当前弧优化的结合大幅减少了搜索空间和无效遍历，多路增广技术则减少了增广次数，使得算法在各类数据集上均能保持高效。

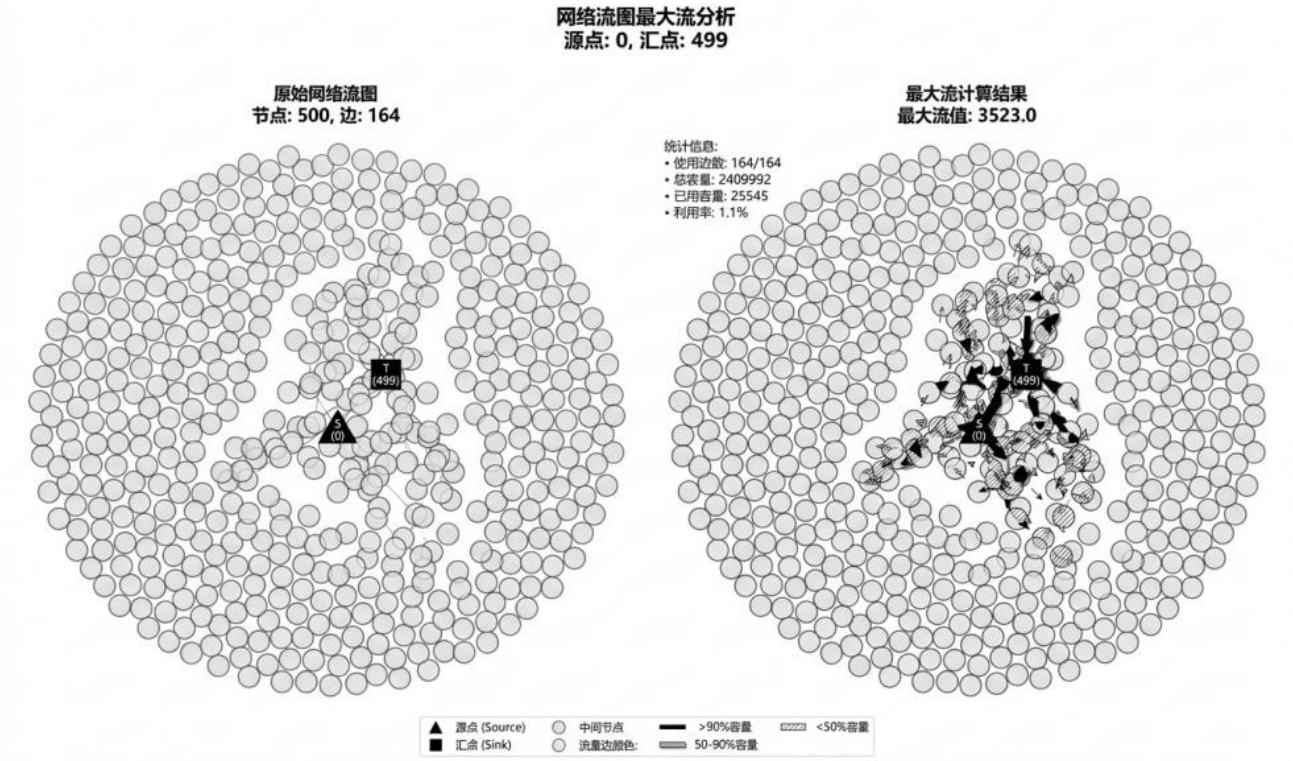

图 2　数据集 B 及其计算结果的可视化图

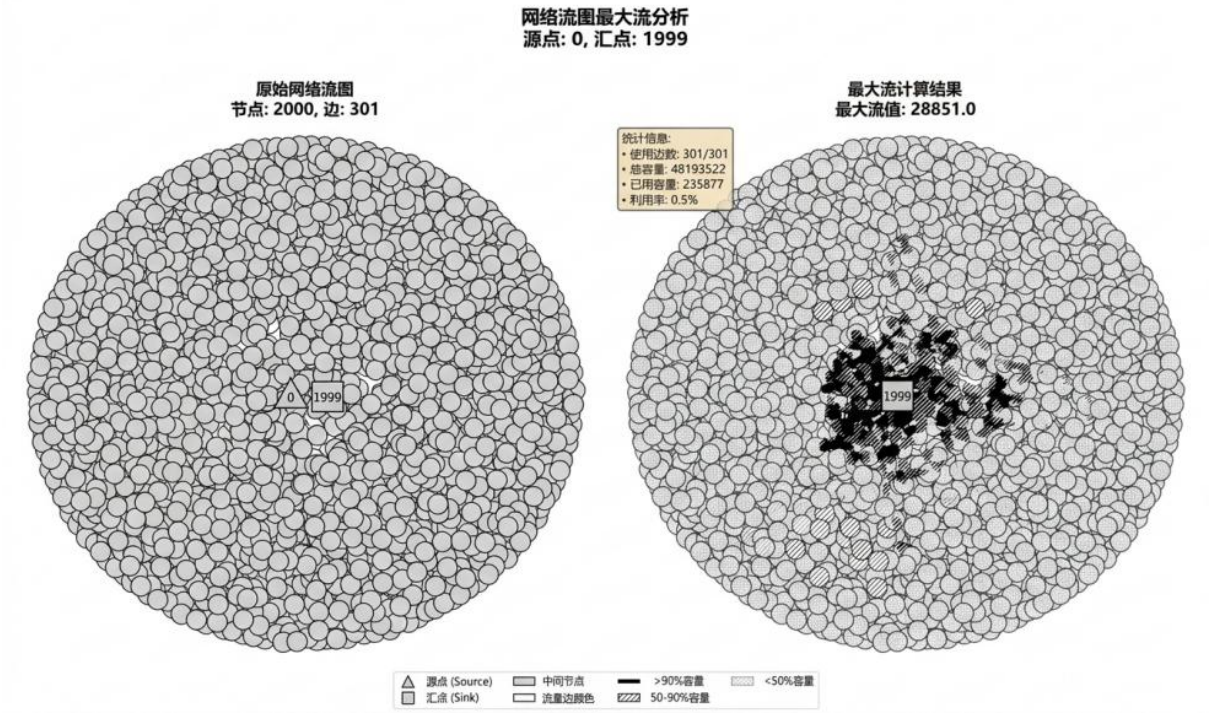

图 3　数据集 C 及其计算结果的可视化图

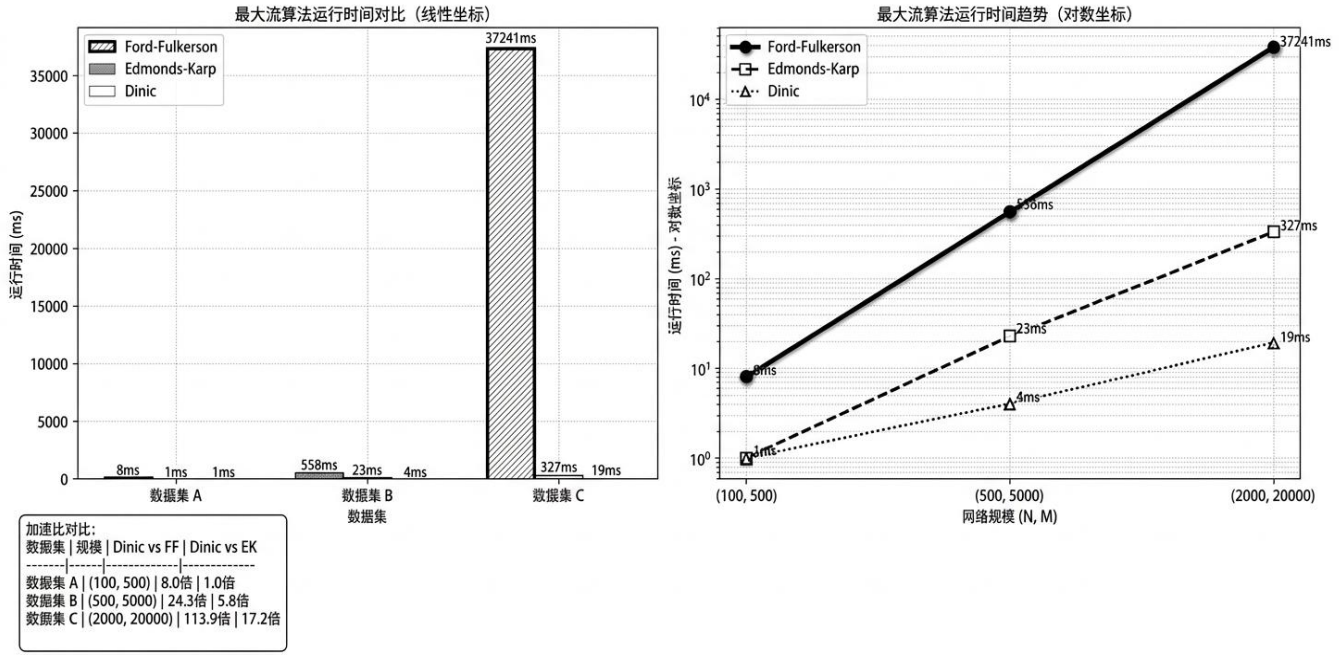

图 4　运行时间随网络规模变化的趋势图

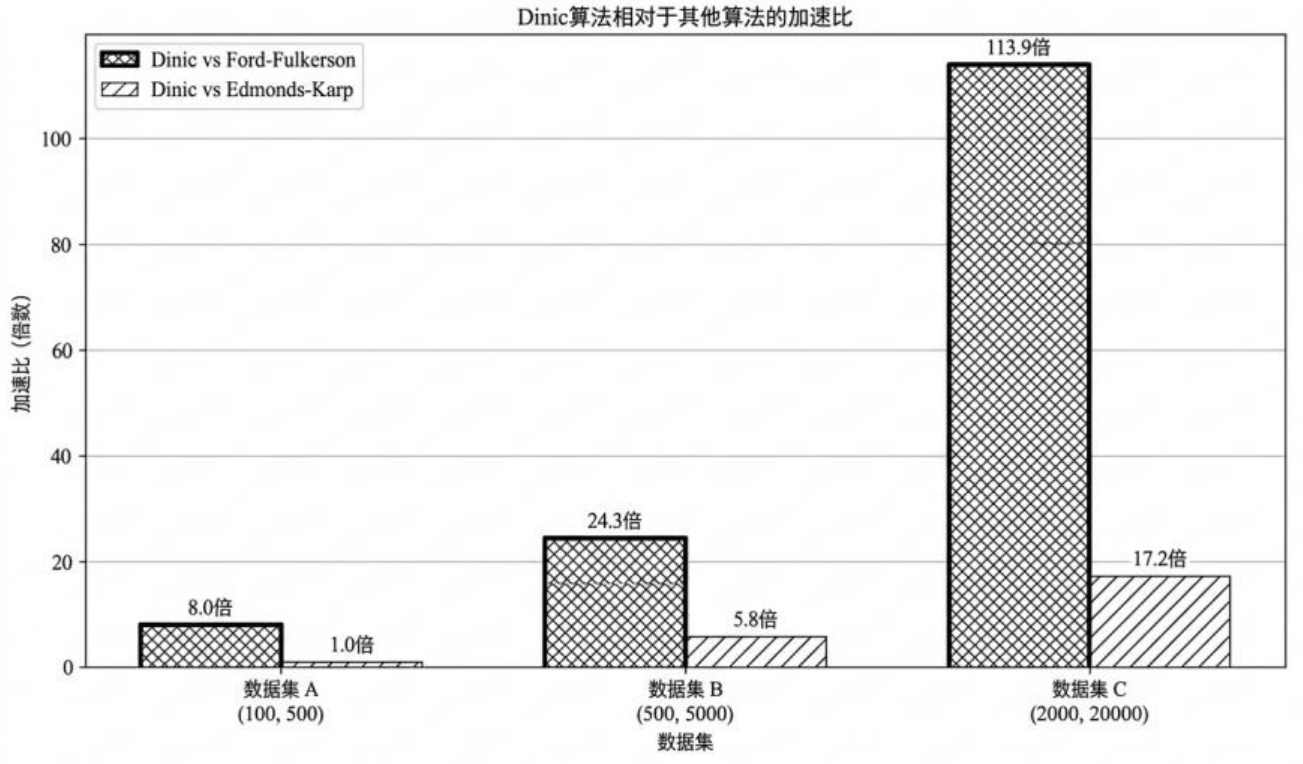

图 5　Dinic 算法相对于其他算法的加速比图

5.5 优化技术的有效性分析

最短增广链策略（Edmonds-Karp）的作用：对比 Ford-Fulkerson 算法，Edmonds-Karp 算法通过 BFS 寻找最短增广链，显著减少了增广次数。

分层图优化的作用：Dinic 算法的分层网络限制了搜索方向，避免了同层节点间的无效搜索和回流。对比 Edmonds-Karp 算法，Dinic 算法在每次 BFS 分层后，可通过一次 DFS 完成多路增广，减少了 BFS 的调用次数。

当前弧优化的作用：当前弧优化通过跳过无效边，减少了 DFS 的遍历开销。在稠密图数据集 B 和 C 中，该优化的效果尤为显著。例如，在数据集 C 中，Dinic

综合优化效果：Dinic 算法结合了分层图、当前弧优化和多路增广技术，在所有数据集上均表现出最优性能。随着网络规模的增大，其加速比逐渐提升，在大规模数据集 C 中，相对于 Ford-Fulkerson 算法的加速比达到 113.9 倍，相对于 Edmonds-Karp 算法的加速比达到 17.2 倍，充分证明了综合优化策略的有效性。

5.6 算法使用场景分析

Ford-Fulkerson 算法：仅适用于小规模、简单拓扑的网络，或对实时性要求极低的场景。由于其效率低下且稳定性差，在实际工程中几乎不被采用；

Edmonds-Karp 算法：适用于中小规模网络，尤其是稀疏图，其实现简单、稳定性好，在对效率要求不高的场景中具有一定的应用价值；

Dinic 算法：适用于大规模、复杂拓扑的网络，包括稠密图、动态网络等对效率和实时性要求较高的场景。例如，在 LEO 卫星网络、大规模物流调度等领域，Dinic 算法能够快速完成最大流计算，为决策提供及时支持。

6 总结与展望

6.1 总结

本文通过三次代码迭代，系统实现了 Ford-Fulkerson、Edmonds-Karp 和 Dinic 三种具有代表性的网络最大流算法，并通过多组实验全面对比了其性能表现。研究结果表明：

搜索策略的优化是提升算法效率的关键。从 DFS 到 BFS（最短增广链）的转变，有效避免了算法在病态数据下的性能退化，将时间复杂度从伪多项式级提升至多项式级；

图结构优化能够进一步缩小搜索空间。分层网络的引入规范了流量的流向，避免了无效回流和同层搜索，显著提升了搜索效率；

当前弧优化（记忆化搜索）是减少计算冗余的核心技术。通过记录节点的当前处理边，避免了对饱和边和无效边的重复遍历，尤其在稠密图中效果显著；算法的运行时间仅为 Edmonds-Karp 算法的 5.8%。除了多路增广的贡献外，当前弧优化减少的重复遍历开销是重要原因；

融合分层图、当前弧优化和多路增广技术的 Dinic 算法，在各类测试场景中均表现出最优性能，其加速比随网络规模增大而显著提升，是处理大规模网络流问题的高效解决方案。

6.2 展望

尽管本文实现的 Dinic 算法表现出优异的性能，但在面对超大规模动态网络（如 LEO 卫星网络、全球物流网络）时，仍存在一定的优化空间。未来的研究方向可以集中在以下几个方面：

探索更高效的分层策略：目前的 BFS 分层仅考虑边数，未考虑边容量，未来可研究基于容量加权的分层策略，进一步优化分层网络的结构，提升增广效率；

动态网络流算法研究：现有算法均针对静态网络设计，当网络拓扑或边容量发生动态变化时，需要重新计算最大流，开销较大。未来可研究增量式 Dinic 算法，通过局部更新分层网络和残留容量，实现动态网络的高效最大流计算；

与其他算法的融合：预流推进算法（如 HLPP 算法）在某些大规模稀疏图中具有更优的理论复杂度。未来可探索将 Dinic 算法的分层策略与预流推进算法的推送重贴标签机制相结合，构建混合式最大流算法，进一步提升算法的适用范围和效率；

并行化实现：随着多核处理器和分布式计算的发展，可研究 Dinic 算法的并行化实现方案，通过多线程并行构建分层网络、并行搜索增广路径等方式，充分利用硬件资源，提升大规模网络流的计算速度。

参 考 文 献

[1] 林俊余,朱磊.基于记忆化搜索的分层网络最大流算法[J].计算机系统应用,2023,32(06):140-148.

[2] Sheng J ,Guan X ,Yang F , et al.An Accelerated Maximum Flow Algorithm with Prediction Enhancement in Dynamic LEO Networks. [J]. Sensors(Basel,Switzerland),2025,25(8)

[3] 邵丽萍.网络最大流算法的研究[D].南京邮电大学,2019.

[4] 周青,杨剑兰.基于有权图的网络最大流标号算法的研究与实现[J].电子技术与软件工程,2023,(02):9-12.

[5] 罗甜甜.基于顶点的网络最大流求解算法[D].南京邮电大学,2020.