计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(6):140−148 [doi: 10.15888/j.cnki.csa.009121]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于记忆化搜索的分层网络最大流算法①

林俊余, 朱　磊

(岭南师范学院 计算机与智能教育学院, 湛江 524048)
通信作者: 朱　磊, E-mail: ll_zhu@163.com

摘　要: 当前, 路由选择算法、计算机视觉图像切割以及机器学习领域的许多问题都可以归结为求解网络最大流. 为了提高基于分层网络最大流算法的效率, 提出了一种基于记忆化搜索策略的最大流算法, 针对传统 Edmonds-Karp 算法和 Dinic 算法重复搜索无效路径所导致的额外开销问题, 设计了一种能够记录搜索状态的记忆化搜索策略, 来避免重复搜索流网络中的无效部分. 实例分析表明了记忆化搜索策略的高效性与可行性. 最终实验结果表明, 基于记忆化搜索的最大流算法执行效率优于传统的 Dinic 算法.

关键词: 最大流; 流网络; 层次网络; 记忆化搜索; 最短增广链路

引用格式: 林俊余,朱磊.基于记忆化搜索的分层网络最大流算法.计算机系统应用,2023,32(6):140–148. http://www.c-s-a.org.cn/1003-3254/9121.html

Maximum Flow Algorithm of Hierarchical Network Based on Memory-aided Search

LIN Jun-Yu, ZHU Lei

(School of Computer Science and Intelligence Education, Lingnan Normal University, Zhanjiang 524048, China)

Abstract: The rooting algorithms, image segmentation in computer vision, and many problems in machine learning can be regarded as problems seeking solutions to the maximum flow of networks. For more efficient maximum flow algorithms based on hierarchical networks, a maximum flow algorithm based on a memory-aided search strategy is put forward. The traditional Edmonds-Karp algorithm and Dinic's algorithm suffer from extra overhead due to repeated searches of invalid paths. Hence, a memory-aided search strategy that can record search states is proposed to conquer this problem. Experimental results show that the proposed strategy is efficient and feasible, and the proposed algorithm outperforms Dinic's algorithm.

Key words: maximum flow; flow network; hierarchical network; memory-aided search; shortest augmenting path

图论问题在计算机科学领域具有重要地位, 作为运筹学领域的一个重要分支, 当今最大流算法已经被广泛运用于实际工程与技术中. 例如, 计算机网络中基于最大流的路由选择算法[1,2], 分布式存储系统的查询[3], 除此之外, 最大流算法还大量运用于计算机视觉领域中对图形的分割、立体和重塑[4,5], 以及机器学习领域[6,7]. 因此, 为了提升对大数据量最大流模型的求解速度, 对最大流算法的优化是十分必要的, 最大流问题的研究具有较高的实用价值.

最大流问题是指在一张给定流网络上进行搜索、计算由源点 s 出发到汇点 t 的最大可行流量. 最大流算法大体上分为两大类: 第 1 类是由 Ford 等人在 1956 年提出的增广算法[8], 增广算法将网络中所能找到的任意一份流量称为可行流量, 当且仅当其符合容量限制与流守恒定律. 该类算法的思想是在流网络中不断地寻找增广路径与该路径的阻塞流, 直到网络中由源点

① 基金项目: 广东省教育厅普通高校特色创新项目 (2021KTSCX065)
收稿时间: 2022-09-30; 修改时间: 2022-10-27, 2022-12-23; 采用时间: 2023-01-13; csa 在线出版时间: 2023-04-25
CNKI 网络首发时间: 2023-04-26

s 到汇点 t 的所有可行路径都被阻塞, 即网络中不存在任何一条增广路径时, 算法终止. 第 2 类最大流算法是由 Karzanov 在 1974 年提出的基于“预流”概念的预流推送算法[9].

对于基于增广路径的经典算法 Ford-Fulkerson 算法而言, Edmonds 等人[10] 和 Dinitz[11] 分别在 1972 年和 1970 年提出了运用层次网络来对搜索策略进行改进. 随着大数据时代的来临, 信息数据量爆炸式激增, 传统的增广路径最大流算法的搜索策略无法高效的应用于现今复杂多样的流网络结构. 因此, 许多学者已经对最大流算法提出了优化和改进, 其中包括对动态网络最大流算法的优化, 对包含交叉顶点的最大流算法优化以及通过构建反向有效网络来优化 Dinic 算法等[12–17].

在实际工程应用中, 传统基于深度优先策略的 Ford-Fulkerson 算法和基于宽度优先策略的 Edmonds-Karp 算法时常因为无法高效的应用于稠密网络而不受工程实践者的青睐, 而基于层次网络的 Dinic 算法有着简洁、易懂且高效的特点, 常受到实践者们的喜爱. 但 Dinic 算法在实际运用的过程中会反复搜索网络的“无效部分”, 且随着流网络规模及稠密程度的增大, Dinic 算法的效率将变得十分低下. 针对这一问题, 本文对基于层次网络的原始 Dinic 算法在增广路径搜索策略上实现优化, 通过记录单次搜索的搜索状态来实现记忆化搜索, 能够有效地避免算法进行无效的搜索和增广, 最终提高算法效率. 除此之外, 在本文实验部分提出了搜索步数、增广步数以及有效搜索率等概念来衡量、量化和对比算法额外代价与搜索效率.

1 基本概念与定义

1.1 最大流的数学模型

若带权有向网络 $G=(V, E, W)$ 满足以下两个条件: 1) s、t 均属于结点集合 V 且源点 s 入度为 0, 汇点 t 出度为 0; 2) 设 $c(u, v)$ 为边容量, 对任意的边满足:

$$\forall (u,v)\in E,\ c(u,v)=W(u,v) \tag{1}$$

则称 G 为流网络, 记作 $G=(V, E, C)$. 在流网络 $G=(V, E, C)$ 中, 设实值函数 f 为 G 的流, 对任意的边满足容量限制:

$$\forall (u,v)\in E,\ 0\leqslant f(u,v)\leqslant c(u,v) \tag{2}$$

且同时满足流守衡定律:

$$\sum\nolimits_{v\in V\backslash u} f(v,u)=\sum\nolimits_{v\in V\backslash u} f(u,v) \tag{3}$$

则称流 f 为流网络 G 上的一个可行流量. 则由流 f 所诱导的残流网络为$R=\left(V,E_f,C_f\right)$, 其中C_f为 R 中边残余流量的集合, 边残余流量为:

$$c_f(u,v)=\begin{cases} c(u,v)-f(u,v),\ (u,v)\in E \\ f(v,u),\ (v,u)\in E \end{cases} \tag{4}$$

其中, 对 R 中任意的边满足:

$$E_f=\left\{(u,v)\in E|\forall c_f(u,v)\geqslant 0\right\} \tag{5}$$

最大流问题是求解当前流网络 G 中的最大可行流 f 的容量总和:

$$f_{\max}=Max\left(\sum\nolimits_{v\in V\backslash\{t\}} f(u,t)\right) \tag{6}$$

1.2 层次网络

层次网络$D=(s,t)\in R$, 记录残流网络 R 中各结点层次, 其中源点 s 在首层, 汇点 t 在末层, 当前残流网络 R' 划分得层次网络D'中不包含 t, 即$D'=(s,k)$, $k\neq t$时认为当前残流网络R'已经不可再被分层, 即此时无法在残留网络中由 s 搜索到 t.

1.3 动态堆栈与已访问结点集

STACK 表示在搜索过程中动态变化的堆栈, 按搜索次序依次存入流网络 G 中各待处理的结点. *SEEN* 是一个无序的结点集合, 保存当前搜索已访问结点.

1.4 深度优先搜索树

$TREE=(V',E')$表示由源点 s 起始的深度优先搜索树, 以多个有序对 (v, u) 的形式表示 DFS 搜索过边 (u, v), 其中 u 是 v 的前驱结点. 当搜索到汇点 t 时, 可以由汇点 t 不断回溯直至源点 s, 即:

$$TREE(s,t)=\{(t\text{-}v),(v\text{-}\cdots\text{-}u),(u\text{-}s)\} \tag{7}$$

1.5 *s-t* 路径上/下连通性

将一条 s-t 路径 P 以结点 n 分为上、下两个部分, 即 $P=(s\text{-}\cdots\text{-}n\text{-}\cdots\text{-}t)=(s\text{-}\cdots\text{-}n, n\text{-}\cdots\text{-}t)$, 其中$n\in V/\{s,t\}$. 定义当$c_f(s\text{-}\cdots\text{-}n)>0$时, 则称路径 P 为上连通, 当$c_f(n\text{-}\cdots\text{-}t)>0$时, 则称路径 P 为下连通. 当且仅当$c_f(s\text{-}\cdots\text{-}n)\cdot c_f(n\text{-}\cdots\text{-}t)>0$路径 P 为 s-t 有效路径, 即增广路径.

2 Dinic 算法问题分析

原始 Dinic 算法的思想是先用宽度优先策略在残流网络 R 上生成层次网络 D, 再对层次网络 D 采用深度优先策略搜索增广路径, 直至当前层次网络中无法找到任意一条 s-t 增广路径时, 对当前残流网络重新划分层次, 直至残流网络无法再生成层次网络时, 算法结束. Dinic 算法描述如算法 1.

算法 1. Dinic 算法

输入: $G=(V, E, C)$, 源点 s 和汇点 t
输出: 残流网络 R 和最大流 $f_{\max}$

1) 初始化: $R(V,E_f,C_f)=G(V,E,C)$, $D=\varnothing$, $f_{\max}=0$;
2) 步骤 1. 使用宽度优先搜索对残流网络 R 分层, 生成层次网络 $D=(s,k), k\in V$, 转步骤 2;
3) 步骤 2. 若汇点 t 属于层次网络D, 则转步骤 3; 反之, 返回当前最大流 $f_{\max}$, 算法结束;
4) 步骤 3. 在所得层次网络D采用深度优先策略搜索 s-t 增广路径 P. 若路径 P 存在, 则转步骤 4; 反之, 转步骤 1;
5) 步骤 4. 计算路径 P 的阻塞流:

$$c_f(P)=\min\left(R(u,v)|\forall(u,v)\in P\right)$$

转步骤 5;
6) 步骤 5. 执行推流, 更新残流网络:

$$R(u,v)=R(u,v)-c_f(P), \forall(u,v)\in P$$

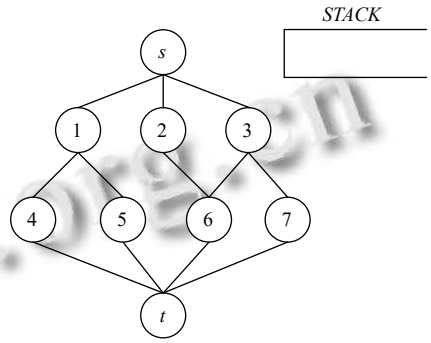
$$R(v,u)=R(v,u)+c_f(P), \forall(u,v)\in P$$

转步骤 6;
7) 步骤 6. 更新最大流$f_{\max}=f_{\max}+c_f(P)$, 转步骤 3;

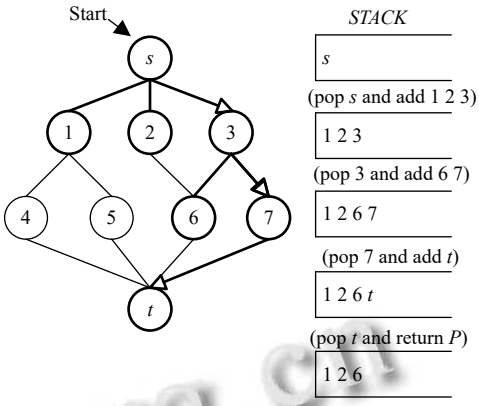
2.1 原始 Dinic 算法与深度优先策略问题分析

原始 Dinic 算法在层次网络中搜索增广路径时, 会随着搜索次数的增加逐渐减慢算法在单张层次网络上的收敛速度, 其原因是深度优先搜索策略总是从源点 s 起始, 每当成功找到 s-t 路径时, 其阻塞流几乎无可避免地会对后续若干次路径搜索产生影响. 在后续搜索中, 深度优先的搜索策略会继续反复的搜索残流网络中的“无效部分”, 这些“无效部分”是若干次搜索所阻塞的路径或流网络中本来就存在的仅具有上连通性而不具有下连通性的路径的集合. 最坏情况下, Dinic 算法在单张层次网络上每次搜索一条 s-t 增广路径的代价随搜索次数依次递增.

如图 1(a) 所示层次网络为例, 当前层次为 3. 图 1–图 3 展示了深度优先策略的搜索步骤, 其中右侧所示 *STACK* 是搜索过程中动态堆栈. 表 1 汇总了搜索过程及搜索代价.

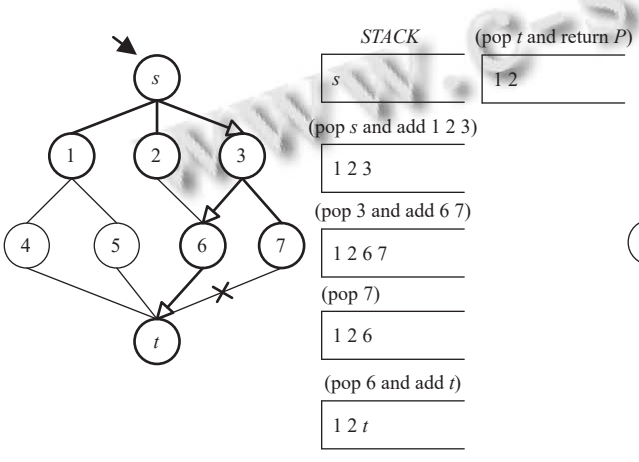


(a) 原始层次网络

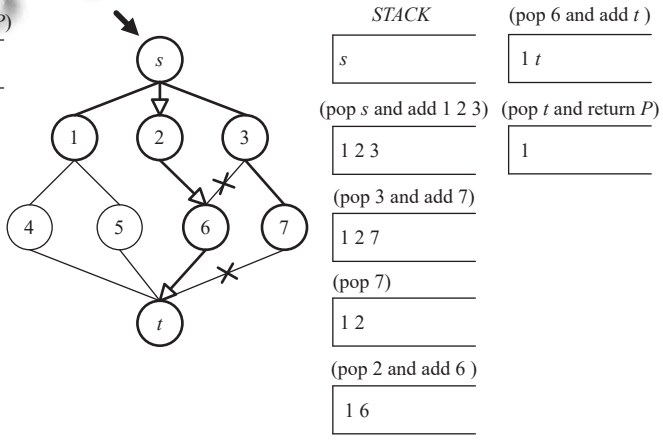


(a) 增广路径 P_1 搜索过程

图 1 原始层次网络与搜索增广路径 P_1

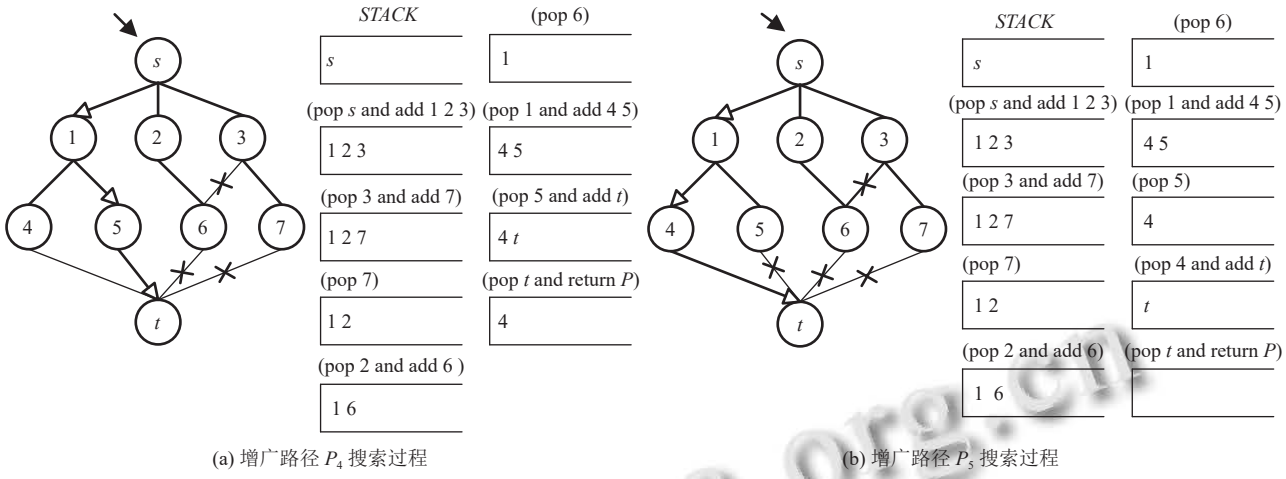


(a) 增广路径 P_2 搜索过程



(b) 增广路径 P_3 搜索过程

图 2 搜索增广路径 P_2、P_3



(a) 增广路径 P_4 搜索过程　　(b) 增广路径 P_5 搜索过程

图 3　搜索增广路径 P_4、P_5

表 1　原始 Dinic 算法搜索过程

搜索路径	增广路径	阻塞边	搜索代价
{s, 1, 2, 3, 6, 7, t}	P_1=(s-3-7-t)	(7, t)	7
{s, 1, 2, 3, 6, 7, t}	P_2=(s-3-6-t)	(3, 6)	7
{s, 1, 2, 3, 7, 6, t}	P_3=(s-2-6-t)	(6, t)	7
{s, 1, 2, 3, 7, 6, 4, 5, t}	P_4=(s-1-5-t)	(5, t)	9
{s, 1, 2, 3, 7, 6, 4, 5, t}	P_5=(s-1-4-t)	(4, t)	9

由上述例子可以得出, 深度优先的搜索策略能够成功地在层次网络中找到所有可能的增广路径 P_1、P_2、P_3、P_4、P_5, 但 Dinic 算法每次都从源点 s 起始, 额外增加了许多不必要的无效搜索步数, 增加额外开销. 例如, 在搜索增广路径 P_2 时, (3, 7) 是无效搜索步, 见图 2(a); 在搜索增广路径 P_5 时, (s, 3)、(3, 7)、(s, 2)、(2, 6)、(1, 5) 是无效搜索步, 见图 3(b). 由此可见, Dinic 算法在层次网络中搜索增广路径的额外代价随搜索次数增加而递增.

3　记忆化搜索算法

3.1　算法思想

由上述举例中不难发现, 相邻的两次路径搜索过程中至少存在前 k 步重合 ($0 \leqslant k \leqslant n$, n 为网络结点数), 仅后 l_P–k 步相异 (l_P 为 P 的路径长度). 例如, 图 2 所示增广路径 P_2、P_3 的搜索过程, 搜索前 5 步 (s, 1)、(s, 2)、(s, 3)、(3, 6)、(3, 7) 重合, 仅最后一步 (7, t) 与 (6, t) 相异. 则考虑是否能记忆化搜索状态, 保证每次仅搜索网络中未被搜索过的部分. 因此, 本文提出了记忆化搜索策略, 且须保证所找到的路径 P 满足以下两点.

(1) 满足连通性限制.

(2) 满足最短路径限制.

深度优先搜索策略每次搜索从源点 s 起始, 仅当 $c_f(u,v)>0,(u,v)\in E'$时会继续向下层搜索, 则可保证所得路径P'的残余流量$c_f(P')>0$, 故 Dinic 算法满足连通性限制. 对于记忆化搜索策略而言, 假设在残流网络 R 中找到一条增广路径P', 其残余流量$c_f(P')=c_f(s\text{-}\cdots\text{-}u,u\text{-}\cdots\text{-}t)=c_f(s\text{-}\cdots\text{-}u)=0$, 若下轮搜索从结点 u 起始得新路径 P, 则会导致 P 的残余流量 $c_f(P)=\min\left(c_f(s\text{-}\cdots\text{-}u),c_f(u\text{-}\cdots\text{-}t)\right)=c_f(s\text{-}\cdots\text{-}u)$=0. 记忆化搜索策略保证了$c_f(u\text{-}\cdots\text{-}t)>0$, 即路径下连通. 但上轮搜索可能会破坏路径的上连通性, 记忆化策略避免从源点 s 开始搜索, 则无法保证算法满足连通性限制. 因此在每轮记忆化搜索开始前, 须对其进行上连通性校验, 仅当$c_f(s\text{-}\cdots\text{-}u)>0$时, 才由结点 u 开始恢复路径搜索, 来保证所得 s-t 路径满足连通性限制, 从而保证路径搜索的有效性.

记忆化搜索策略还需保证所得 s-t 路径满足最短路径限制. 假设层次网络D当前层次为 l, 则在残流网络 R 中不存在路径长度小于 l 的增广路径, 即$\forall P\in R|L(P)\geqslant l$. 若首次记忆化搜索成功找到增广路径$P_1(s\text{-}\cdots\text{-}u,u\text{-}\cdots\text{-}t)$则后续搜索将从结点$u$起始, 其中$u\in V/\{s,t\}$, 设结点 u 属于D中第 h 层($1\leqslant h<l$), 则由首次搜索可以保证$P'_1=(s\text{-}\cdots\text{-}u)$是最短路径, 其路径长度$L\left(P'_1\right)=h$, 而后续搜索由层次网络性质可以保证$P_2=(u\text{-}\cdots\text{-}t)$的路径长度满足$L(P_2)\leqslant l-h$, 故当前 s-t 路径长度满足$L\left(P'_1\right)+L(P_2)\leqslant l$. 综上, 若当前残流网络 R 存在 s-t 增广路径, 则其路径长度必为 l, 即满足最短路径限制.

3.2　算法描述

本文提出的记忆化搜索策略与常规深度优先搜索策略不同之处在于, 算法将上轮搜索结果和状态保存

在 *STACK*, *SEEN*, *TREE* 中, 在此基础上进行下一次搜索并持续更新当前搜索状态, 来保证每次调用搜索时总能针对网络中未搜索过的部分, 避免了原始 Dinic 算法反复的搜索残流网络中的“无效部分”. 从宏观上看, 每轮执行的若干次记忆化搜索等效于在单张层次网络上进行一次完整的深度优先遍历, 得到若干条增广路径. 记忆化搜索算法描述如算法 2.

算法 2. 记忆化搜索算法

输入: $R=(V,E,C')$, D, *STACK*, *SEEN*, *TREE*, 源点 s 和汇点 t
输出: *STACK*, *SEEN*, *TREE*

1) 步骤 1. 若 *STACK* 非空则取栈顶元素 u, 转步骤 2; 反之, 算法结束;
2) 步骤 2. 若所取得顶点 u 是源点 s, 则转步骤 4; 反之, 转步骤 3;
3) 步骤 3. 若结点 u 在搜索树 *TREE* 中上连通性校验通过, 则转步骤 4; 反之, 将结点 u 设置成未访问态, 即$SEEN=SEEN\backslash\{u\}$, 转步骤 1;
4) 步骤 4. 获取结点 u 的邻接结点 v, 转步骤 5;
5) 步骤 5. 若结点 v 是未访问态, 即$v\notin SEEN$, 转步骤 6; 反之, 转步骤 4;
6) 步骤 6. 若邻接结点 v 在结点 u 的下一层, 即$D[v]=D[u]+1$, 转步骤 7; 反之, 转步骤 4;
7) 步骤 7. 将结点 v 加入动态堆栈 *STACK*; 将结点 v 设置成已访问态; 将边(v,u)加入 *TREE*; 结点 v 迭代结点 u, 即$u\leftarrow v$, 转步骤 4.

基于记忆化搜索策略的最大流算法首先会根据残流网络 R 来构建层次网络D, 在层次网络上不断执行记忆化搜索算法来寻找增广路径, 直到当前层次网络无法再被找到任意一条 s-t 增广路径时, 对当前残流网络重新划分层次, 直到所划分的层次网络中不包含 t 时, 算法结束. 此时所找到的增广路径的流量总合为最大流. 算法流程图见图 4. 算法描述如算法 3.

算法 3. 基于记忆化搜索策略的最大流算法

输入: $G=(V, E, C)$, 源点 s 和汇点 t
输出: 残流网络 R 和最大流 $f_{\max}$

1) 初始化: $R(V,E_f,C_f)=G(V,E,C)$, $f_{\max}=0$, $D=\varnothing$, $STACK=\varnothing$, $SEEN=\varnothing$, $TREE=\varnothing$;
2) 步骤 1. 使用宽度优先搜索对残流网络 R 分层, 生成层次网络 $D=(s,k),k\in V$, 转步骤 2;
3) 步骤 2. 若汇点 t 属于层次网络 D, 转步骤 3; 反之, 返回当前最大流 $f_{\max}$, 算法结束;
4) 步骤 3. 将源点 s 加入动态堆栈 *STACK* 中, 即 $STACK=\{s\}$; 将源点 s 设置成已访问态, 即$SEEN=\{s\}$, 转步骤 4;
5) 步骤 4. 在所得层次网络 D 上采用记忆化搜索策略搜索 s-t 增广路径 P. 若路径 P 存在, 则转步骤 5; 反之, 转步骤 1;
6) 步骤 5. 计算路径 P 的阻塞流:

$$c_f(P)=\min\{R(u,v)|\forall(u,v)\in P\}$$

转步骤 6;
7) 步骤 6. 执行推流, 更新残流网络:

$$R(u,v)=R(u,v)-c_f(P),\forall(u,v)\in P$$

$$R(v,u)=R(v,u)+c_f(P),\forall(u,v)\in P$$

转步骤 7;
8) 步骤 7. 更新最大流$f_{\max}=f_{\max}+c_f(P)$; 将汇点 t 设置成未访问态, 即$SEEN=SEEN\backslash\{t\}$, 转步骤 4.

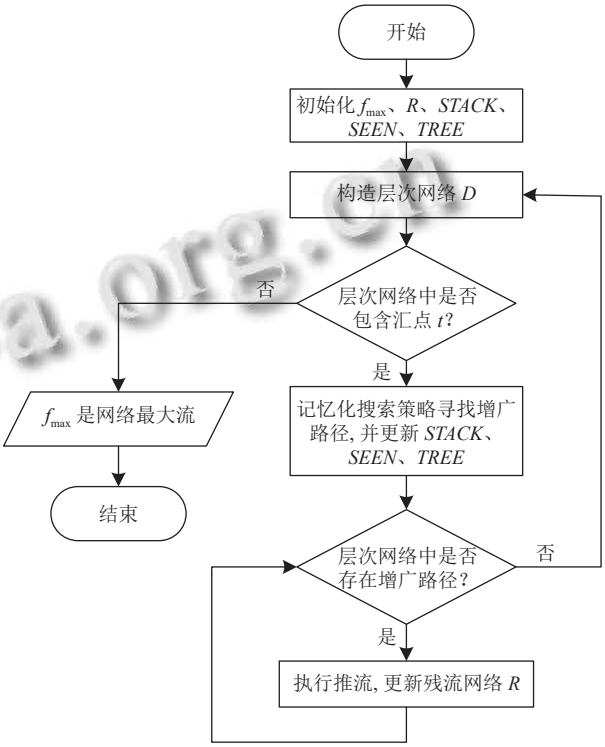


图 4 基于记忆化搜索策略的最大流算法流程图

本文所提出的基于记忆化搜索策略的最大流算法, 须在层次网络上执行搜索. 采用何种搜索策略不直接影响算法何时停止搜索增广路径, 算法的终止条件是当前层次网络中不包含汇点 t. 因此, 若所采取的搜索策略能在当前层次网络中找到至少一条 s-t 增广路径, 则一定能在后续若干张层次网络中找到最大流.

当采用记忆化搜索策略时, 设所找到的 s-t 增广路径为 P, 则每轮搜索会将至少 l_P–1 (l_P 为 P 的路径长度) 个结点的访问状态标记成已访问态, 并且每轮搜索结束后仅把汇点t从*SEEN*中移除, 从而保证搜索路径不重叠. 但可能会导致某些路径被遗漏, 例如, Dinic 算法在如图 1(a) 所示在层次网络中能找到全部的 5 条 s-t 增广路径, 而本文提出的算法无法找到由图 2(b) 所示增广路径P_3 = (s-2-6-t), 因为记忆化的策略在搜索路径 P_2 时已经将结点 6 标记为已访问态, 故只能在下一张层次网络中找到 P_3. 对于如图 5 所示层次网络中的 k 条增广路径, 则需要通过 k 张层次网络才能被全部找到.

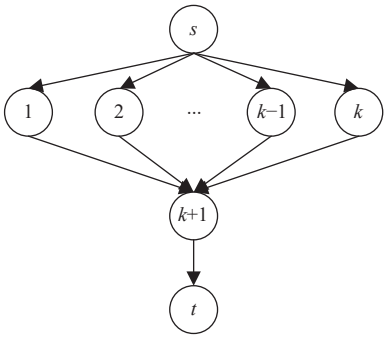
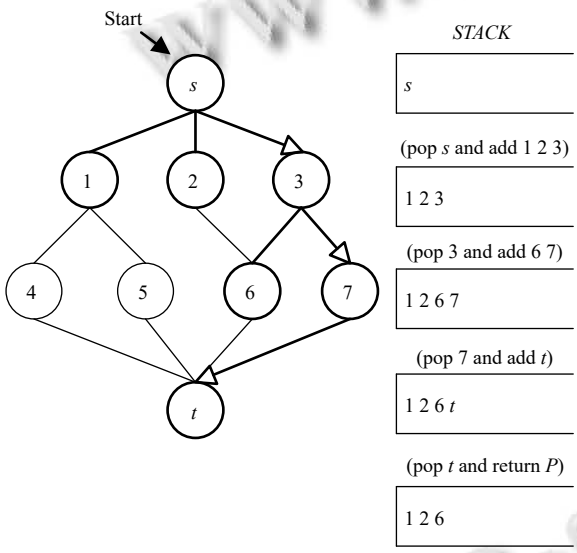


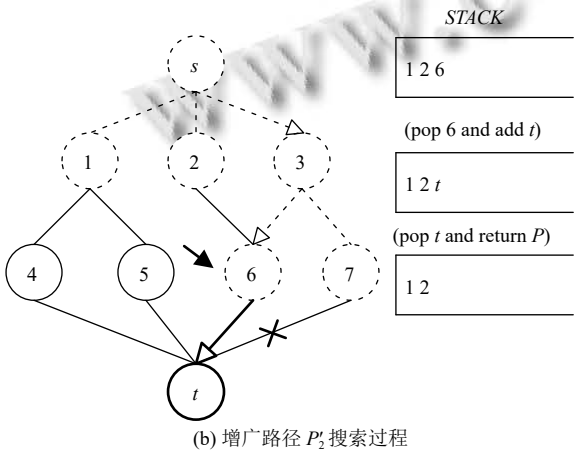
图 5 记忆化搜索策略的最坏情况

3.3 实例分析

初始层次网络如图 1(a) 所示, 当前层次为 3. 图 6、图 7 展示了记忆化搜索策略的搜索步骤, 右侧所示 *STACK* 是搜索过程中动态堆栈, 显示了搜索次序和过程. 图 6、图 7 用虚线标识网络中已搜索过的部分. 表 2 汇总了记忆化策略的搜索过程及搜索代价.

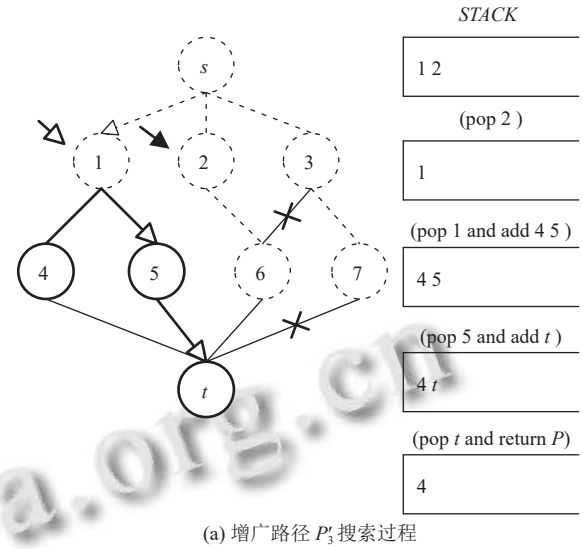


(a) 增广路径 P'_1 搜索过程

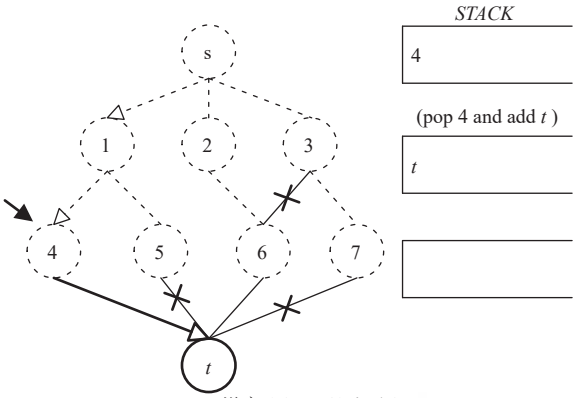


(b) 增广路径 P'_2 搜索过程

图 6 搜索增广路径 P'_1、P'_2



(a) 增广路径 P'_3 搜索过程



(b) 增广路径 P'_4 搜索过程

图 7 搜索增广路径 P'_3、P'_4

表 2 记忆化策略搜索过程

搜索路径	增广路径	阻塞边	搜索代价
{*s*, 1, 2, 3, 6, 7, *t*}	P'_1=(*s*-3-7-*t*)	(7, *t*)	7
{6, *t*}	P'_2=(*s*-3-6-*t*)	(3, 6)	2
{1, 4, 5, *t*}	P'_3=(*s*-1-5-*t*)	(5, *t*)	4
{4, *t*}	P'_4=(*s*-1-4-*t*)	(4, *t*)	2

记忆化策略首次搜索与深度优先策略保持一致, 算法依次遍历结点 (*s*, 1, 2, 3, 6, 7, *t*) 后找到增广路径 P'_1 = (*s*-3-7-*t*), 搜索代价为 7, 本次搜索状态将被保留. 搜索过程见图 6(a).

第 2 次搜索从结点 6 开始, 假设层次网络中边 (7, *t*) 被阻塞, 算法通过 (6, *t*) 找到汇点 *t*, 搜索所得增广路径为 P'_2 = (*s*-3-6-*t*), 搜索代价为 2. 搜索过程见图 6(b).

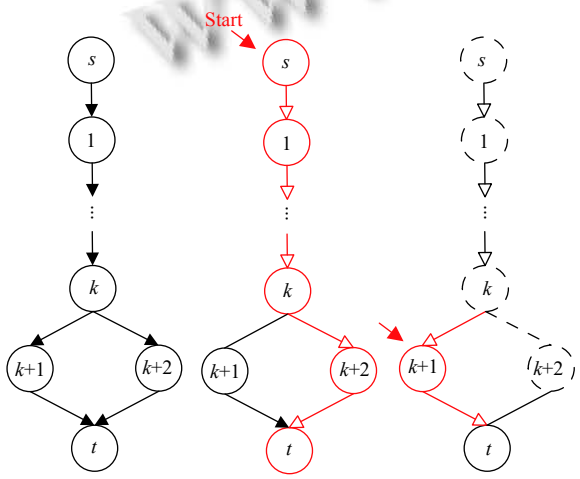
第 3 次搜索从结点 2 开始, 假设层次网络中边 (7, *t*)、(3, 6) 被阻塞, 由于结点 6 已经在上轮搜索中已被标记

成已访问态, 故算法无法通 (2, 6, t) 找到汇点 t. 因此, 算法将从结点 1 继续恢复搜索, 依次遍历结点 (4, 5, t) 找到汇点 t, 所得增广路径为P'_3 = (s-1-5-t), 本次搜索代价为 4. 搜索过程见图 7(a).

根据上一轮搜索所保存的搜索状态, 最后一轮搜索从结点 4 开始, 假设层次网络中边 (7, t)、(3, 6)、(5, t) 被阻塞, 算法通过 (4, t) 找到汇点 t, 所得增广路径为 P'_4 = (s-1-4-t), 本次搜索代价为 2. 搜索过程见图 7(b).

3.4 复杂度分析

若当流网络中存在类似图 8(a) 所示的带状分支结构, 两条增广路径长度同为 k+2, 则路径有效搜索步数 Vn为$2\cdot(k+2)$. 如图 8(b)、图 8(c) 所示, 记忆化策略所得两条路径的搜索步数分别为$S_{n1}=k+2, S_{n2}=2$; 则有效搜索率$Vr=Vn/(S_{n1}+S_{n2})=2(k+2)/(k+4)>1, k>0$, 故有效搜索率 Vr 可能大于 1.



(a) 带状分支结构 (b) 第 1 次增广 (c) 第 2 次增广

图 8 记忆化搜索策略的最优情况

在包含 n 个结点、m 条边的流网络 G 中, 最坏情况下, 每张层次网络中仅能找到一条增广路径, 算法效率等同于基于宽度优先搜索策略的 Edmonds-Karp 算法, 因此最坏复杂度是O(nm^2). 最优情况下, 每张层次网络中都包含若干条 s-t 路径, 而 G 中 n 个结点所能构成的最长 s-t 路径长度为 n−1, 故流网络 G 至多能够划分出 n 张层次网络, 每次构建层次网络的代价为O(m). 记忆化策略搜索增广路径的代价不超过O(n). 因此, 在一张层次网络上搜索和增广推流的总代价不超过O(nm). 在开始搜索前还需要对结点进行上连通性校验, 代价不超过O(n). 所以单张层次网络上记忆化搜索的总代价为O($n+m+nm$), 故算法复杂度为O($n(n+m+nm)$) = O(n^2m). 因此, 算法整体复杂度介于O(nm^2)与O(n^2m)之间.

4 实验与仿真

本文实验部分所有测试代码均是基于 Python 实现, 由 Python 3.9 编译, 实验测试平台配置为 Intel i7 2.59 GHz CPU 和 16 GB 内存, 64-bit Windows 操作系统的计算机, 本文实验所采用的测试数据均为随机生成的流网络, 在对应 ENr 下分别生成 10 张随机流网络进行测试, 测试结果取平均值. 实验变量定义如下.

1) FVm 表示最大流的流量值.

2) ENr 表示随机流网络中边数与结点数的比值.

3) Vn 表示有效搜索总步数, 即增广路径长度总和; Sn 表示搜索总步数, 即搜索总代价; Vr 是有效搜索率, 其定义为: $Vr=Vn/Sn$.

本文首先在数值上验证了算法的正确性, 在算法执行结束后, 将结合最大流值 FVm、残流网络 R、流网络 G 来计算流经每一条边的流量值, 计算其与本文算法所得最大流量值、源点 s 发出的流量值以及流入汇点 t 的流量值是否相等, 来验证最大流量 FVm 数值大小的正确性. 此外, 还将验证每个结点 (除 s、t 以外) 的存余流量 (excess flow) 均为 0, 来确定本文算法所得最终残流网络中不存在 s-t 路径. 通过上述步骤, 可以证明本文算法在数值上的正确性与可行性.

本文在初始包含 500 个结点、5 000 条边, 每次扩张 5 000 条边的随机流网络中, 对比了 3 种传统最大流算法: Ford-Fulkerson 算法、Edmonds-Karp 算法、Dinic 算法以及本文所提的记忆化搜索算法的耗时和搜索效率. 算法耗时的测试结果如图 9 所示, 当流网络较为稀疏时, Ford-Fulkerson 算法耗时上升明显, 是由于深度优先搜索在稀疏流网络中搜索路径的任意性导致算法所得增广路径过长, 进而导致 Ford-Fulkerson 算法耗时开销明显增大. 而在基于层次网络的最短增广算法中, Edmonds-Karp 算法、Dinic 算法以及本文算法在稀疏网络上耗时差距不大. 但随着流网络稠密程度上升, Edmonds-Karp 算法、Dinic 算法的耗时上升明显. 其原因是 Edmonds-Karp 算法所采用的宽度优先搜索策略每轮搜索大量重复, 导致其搜索代价随着网络稠密程度的增加而递增. Dinic 算法通过循环利用层次网络, 减少了宽度优先搜索的调用次数, 因此其算法耗时相较于 Edmonds-Karp 算法有所下降. 本文算法通过记忆

化策略更高效的利用层次网络中的连通路径, 进一步避免了在无效路径上的无效搜索, 因此在稠密网络中的算法耗时大幅下降.

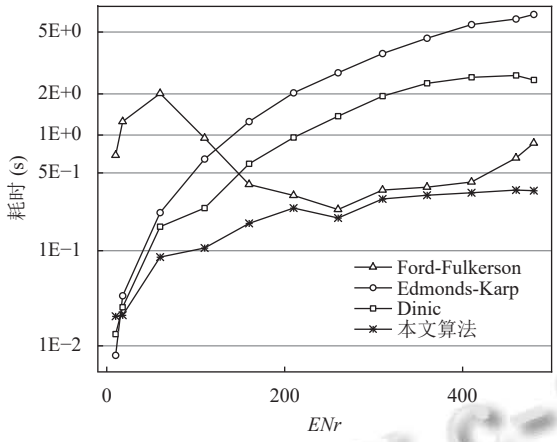


图 9 算法耗时对比

搜索效率的对比主要针对基于层次网络的 4 种最大流算法, 分别是 Edmonds-Karp 算法、Dinic 算法以及逆向 Dinic 算法和本文算法, 测试结果如图 10 和表 3 所示. 逆向 Dinic 算法是由文献 [14] 所提出的基于有效反向网络的最大流算法, 该算法通过不断地剪枝, 剔除$c_f(u,v)=0$的边, 再通过构建反向网络逆向搜索 *t*-*s* 路径, 从而减少搜索代价. 随着随机网络稠密程度增加, 大量无效搜索使得 Edmonds-Karp 算法和 Dinic 算法的有效搜索率 *Vr* 始终低于 5%. 而本文算法的搜索代价变化趋势较为平稳, 在 *ENr* 小于 100 的稀疏流网络中 *Vr* 略高于其余两个传统算法. 在稠密的网络结构中, 记忆化搜索策略对层次网络中连通路径的利用率较高, 有效搜索率提升较为显著.

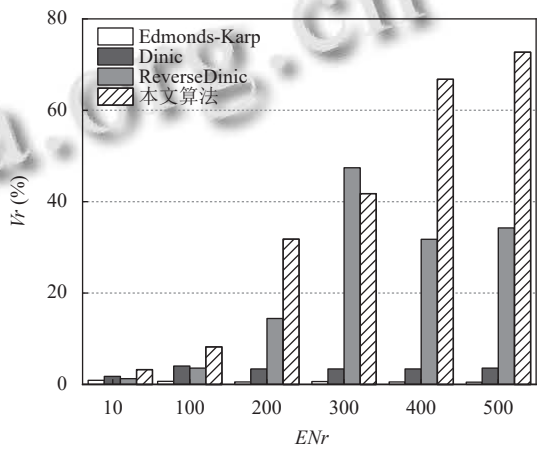


图 10 有效搜索率对比

表 3 搜索代价和有效搜索率测试结果

ENr	Edmonds-Karp		Dinic		ReverseDinic		本文算法	
	Sn	*Vr* (%)	*Sn*	*Vr* (%)	*Sn*	*Vr* (%)	*Sn*	*Vr* (%)
12	11986	0.9	6088	1.8	7615	1.3	3633	3.2
100	118339	0.6	18564	4.0	19566	3.6	7998	8.3
200	229601	0.5	38025	3.4	8709	14.4	3503	31.8
300	312936	0.6	54110	3.4	3904	47.4	4244	41.8
400	404719	0.6	70411	3.4	7837	31.8	3641	66.8
500	510137	0.5	72482	3.6	7543	34.2	3786	72.7

逆向 Dinic 算法的搜索代价及有效搜索率波动较大, 测试结果表明, 其有效搜索率在本文所定义的随机网络结构中难以体现较为平稳的增长趋势. 在测试过程中发现逆向 Dinic 算法往往在上连通性较好的网络中对有效搜索率 *Vr* 的优化明显, 但在上连通性不佳的网络中 *Vr* 接近于原始 Dinic 算法, 其原因是逆向 Dinic 算法在搜索过程中路径递归迭代次数不断增加, 若当前路径不连通时逐层路径回退会增加额外搜索代价, 且剪枝操作至多会增加$O(m)$的额外代价. 因此, 逆向 Dinic 算法在上连通性不佳的网络中有效搜索率较低, 而在上连通性较好的网络中, 逆向搜索策略几乎不会产生额外代价, 从而保证了较好的效率. 记忆化搜索策略始终保持正向搜索避免路径回退, 虽然存在路径遗漏问题会导致搜索循环若干轮, 但每轮额外搜索代价恒定, 故记忆化搜索策略相较于逆向 Dinic 算法对有效搜索率 *Vr* 的优化更为稳定, 且在随机连通流网络中执行效率更高.

5 结论与展望

本文提出了一个基于记忆化搜索策略的最大流算法, 该算法在基于层次网络的最大流算法上优化了搜索效率, 通过记忆化存储路径搜索状态来避免算法反复搜索网络中的“无效部分”, 进而避免了大量无效增广. 实验结果表明, 算法在稠密网络中的搜索总代价与

算法执行效率明显优于传统 Edmonds-Karp 算法和 Dinic 算法, 且比同类算法更具稳定性. 与此同时, 本文算法仍旧保留着与传统增广路径算法一致的简易性与可行性. 现如今, 最大流算法被广泛地运用于实际工程技术当中, 本文算法具有较高的实用价值, 为最大流算法相关工程技术的应用与开发提供了一个便捷、可行且高效的选择.

参考文献

1 Alzaben N, Engels DW. End-to-end routing in SDN controllers using max-flow min-cut route selection algorithm. Procedings of the 24th International Conference on Advanced Communication Technology (ICACT). PyeongChang: IEEE, 2021. 461–467.

2 毛善丽, 李晓卉, 蔡彬, 等. 基于 EHWSN 的能量均衡动态最大流路由算法. 传感技术学报, 2017, 30(2): 291–295. [doi: 10.3969/j.issn.1004-1699.2017.02.021]

3 徐毅, 王建民, 黄向东, 等. 一种基于最大流的分布式存储系统中查询任务最优分配算法. 计算机学报, 2019, 42(8): 1858–1872. [doi: 10.11897/SP.J.1016.2019.01858]

4 Duan YP, Huang WM, Chang HB. Shape prior regularized continuous max-flow approach to image segmentation. Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012). Tsukuba: IEEE, 2012. 2516–2519.

5 Baxter JSH, Rajchl M, Mcleod AJ, *et al*. Directed acyclic graph continuous max-flow image segmentation for unconstrained label orderings. International Journal of Computer Vision, 2017, 123(3): 415–434. [doi: 10.1007/s11263-017-0994-x]

6 Pang SN, Zhu L, Bany T, *et al*. Online max-flow learning via augmenting and de-augmenting path. Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN). Rio de Janeiro: IEEE, 2018. 1–8.

7 Zhu L, Pang SN, Sarrafzadeh A, *et al*. Incremental and decremental max-flow for online semi-supervised learning. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(8): 2115–2127. [doi: 10.1109/TKDE.2016.2550042]

8 Ford LR Jr, Fulkerson DR. Maximal flow through a network. Canadian Journal of Mathematics, 1956, 8: 399–404. [doi: 10.4153/CJM-1956-045-5]

9 Karzanov AV. Determining the maximal flow in a network by the method of preflows. Soviet Mathematics Doklady, 1974, 15(3): 434–437.

10 Edmonds J, Karp RM. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM, 1972, 19(2): 248–264. [doi: 10.1145/321694.321699]

11 Dinitz EA. Algorithm for solution of a problem of maximum flow in networks with power estimation. Soviet Mathematics Doklady, 1970, 11(5): 1277–1280.

12 张柏礼, 王媛瑗, 洪亮, 等. 动态网络中最大流快速增量求解. 东南大学学报 (自然科学版), 2017, 47(3): 450–455. [doi: 10.3969/j.issn.1001-0505.2017.03.006]

13 罗甜甜, 赵礼峰. 包含交叉顶点的最大流改进算法. 计算机工程, 2020, 46(11): 48–52. [doi: 10.19678/j.issn.1000-3428.0056170]

14 韩颖铮, 邓国强, 陆以勤. 基于有效反向网络的最大流算法. 通信学报, 2018, 39(S1): 179–183.

15 赵礼峰, 邵丽萍. 求解最大流问题的算法. 计算机工程与设计, 2019, 40(8): 2224–2227, 2241. [doi: 10.16208/j.issn1000-7024.2019.08.020]

16 邵丽萍, 赵礼峰. 基于宽度优先的网络最大流求解算法. 计算机技术与发展, 2019, 29(6): 62–65. [doi: 10.3969/j.issn.1673-629X.2019.06.013]

17 Wei W, Liu Y, Zhang RQ. SPLMax: Exploiting the simple path introduced locality for maximum flow acceleration. IEEE Communications Letters, 2018, 22(7): 1330–1333. [doi: 10.1109/LCOMM.2018.2830786]

(校对责编: 孙君艳)