深 圳 大 学 实 验 报 告

课程名称 算法设计与分析

项目名称 最大流应用问题

学　　院 计算机与软件学院

专　　业 软件工程

指导教师 卢亚辉

报 告 人 郑彦薇 学号 2020151022

实验时间 2022/6/7~2022/6/21

提交时间 2022/6/19

教务处制

一、实验目的与要求

1. 掌握最大流算法思想。
2. 学会用最大流算法求解应用问题。

二、实验内容与方法

（一） 实验内容

1. 有m篇论文和n个评审，每篇论文需要安排a个评审，每个评审最多评b篇论文。请设计一个论文分配方案。
2. 要求应用最大流解决上述问题，画出m=10，n=3的流网络图并解释说明流网络图与论文评审问题的关系。
3. 编程实现所设计算法，计算a和b取不同值情况下的分配方案，如果没有可行方案则输出无解。

（二）求解方法

根据实验内容知道，我们需要应用最大流解决上述论文--评审分配问题，而求解最大流问题的方法有 Ford-Fulkerson 方法。

（三）方法和相关概念引入

1. Ford-Fulkerson 方法（以下简称 FF 方法）：FF 方法是一种求解最大流问题的迭代方法，它依赖于三种重要思想：残留网络、增广路径和割。
方法基本伪代码：

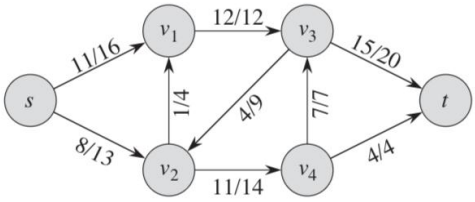
方法 1 Ford-Fulkerson 方法
输入： G 流网络，s 起点，e 汇点

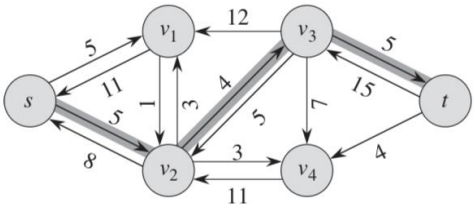
```
1: function F-F(G, s, t)
2:     f ← 0
3:     while exist a path p from s to e do
4:         沿着增广路径 p 增广流 f
5:     end while
6:     return f
7: end function
```

2. 残留网络：残留网络就是有残留容量的图，即在图中增加一条反向边，反向边的值为该边的流的大小，原始边的值为边的容量减去边的流大小。如下公式所示：

$$c_f(u,v)=\begin{cases} c(u,v)-f(u,v), if\,(u,v)\in E, \\ f(u,v), if\,(v,u)\in E, \\ 0, otherwise. \end{cases}$$

通过下图进行进一步说明，左图所示的流网络对应的残留网络如右图所示：





3. 增广路径：增广路径是指在一个残留网络中，一条从源点 s 到汇点 e 的简单路径。当残留网络确定时，我们可以通过 BFS 或 DFS 求得增广路径，这也是问题求解的关键。

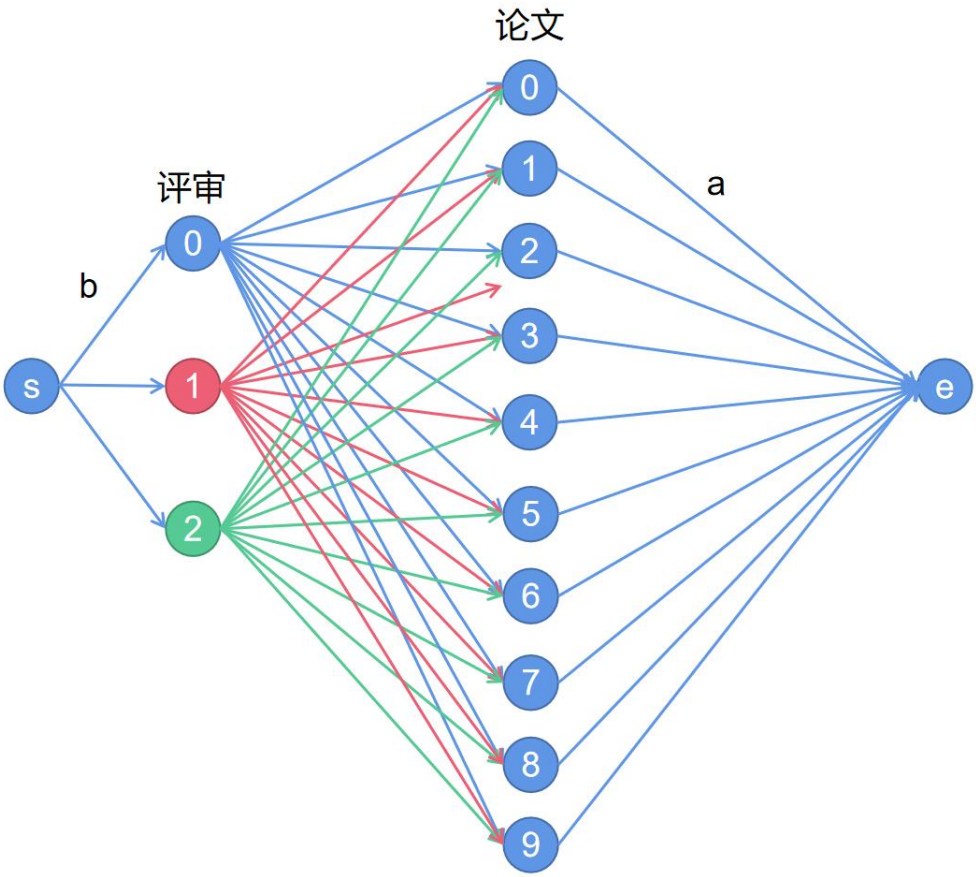
4. 最大流最小切割定理：该方法告诉我们，一个流是最大流当且仅当它的残留网络中不包含增广路径。这个定理是求解问题的基础。

三、实验步骤与过程

（一）问题转换

根据上面所提相关概念和方法，我们可以知道解决论文和评审的分配问题，可以将该问题转化为最大流问题。

已知对于 m 篇论文，n 位评审，规定一篇论文需要安排 a 个评审，一个评审最多可以评 b 篇论文。将论文和评审转换为节点，将评审可以处理的论文数 b 作为评审的入度，将每个论文需要安排的评审数作为论文的出度，加上首尾的起点 s 和汇点 e，构造该问题的流网络如下图所示：



对于 a 个评审，总共需要处理的论文数为 a • m 篇，因此判断是否存在可行的分配方案则是

判断所求的最大流等不等于 am，若等于，说明存在解决方案；若不等于，说明不存在可行的分配方案。

综上，我们可以得到解决问题的整体思路为：得到流的残留网络→使用 DFS 或 BFS 获得残留网络中的增广路径→根据增广路径是否存在判断流是否是最大流→若这个流为最大流（增广路径不存在），判断最大流与 am 是否相同→得到结果。

（二）FF 基本法

1. 解题思路

1) 根据上述问题转换，首先根据输入存储图信息，包括源点到评审、评审到论文、论文到汇点的边信息以及权值。其中源点到评审的权值为评审最多可以评审的论文数 b，评审到论文的权值为 1，论文到汇点的权值为评一篇论文需要的评审数 a。
2) 使用 DFS 寻找从源点到汇点的增广路径，当残留网络中还存在增广路径时，记录下当前 DFS 返回的流量值，继续通过 DFS 查找是否还存在增广路径。若增广路径不存在（即 DFS 返回值为 0），则返回当前总流量值即为最大流。
3) 记录通过上述 DFS 递归查找得到的最大流的值，与论文数和论文所需评审数的乘积（即 a*m）进行比较，如果该值可以等于 am（即 am<=最大流），则存在分配方案；否则不存在。

2. 伪代码

根据思路，编写伪代码如下：

算法 1 Ford-Fulkerson

输入: G 流网络，s 起点，f 流

```
1: function DFS(start, f)
2:     if start == vexnum then
3:         return f
4:     end if
5:     for i = 1 → vexnum do
6:         if Map[start][i] > 0 and visit[i] == false then
7:             visit[i] ← true
8:             temp ← DFS(i, min(f, Map[start][i]))
9:             if temp > 0 then
10:                Map[start][i] −= temp
11:                Map[i][start] += temp
12:                return temp
13:            end if
14:        end if
15:    end for
16:    return f
17: end function
18:
19: function FF(start)
20:    f ← 0
21:    while true do
22:        visit[] ← 0
23:        ans ← DFS(start, INF)
24:        if ans == 0 then                ▷ 残留网络不存在增广路径，得到最大流
25:            return f
26:        end if
27:        f += ans                        ▷ 找到一个流量为 ans 的增广路径
28:    end while
29: end function
```

3. 算法正确性验证

对于 10 篇论文、3 个评委，将 a 赋值为 1，b 赋值为 5。假设评审编号为 A、B、C，论文编号从 0~9，可以任意得到一种分配方案为：A 负责 0~4，B 负责 5~9.（存在多种方案，这里只验证是否有分配方案的判断的正确性）。

输入相应的值，得到结果如下：

```
请输入论文数量m: 10
请输入评审数量n: 3
请输入一篇论文需要安排的评审数a: 1
请输入一个评审最多可评论文数b: 5
The MaxFlow is 10
存在分配方案
the time cost is: 0ms
```

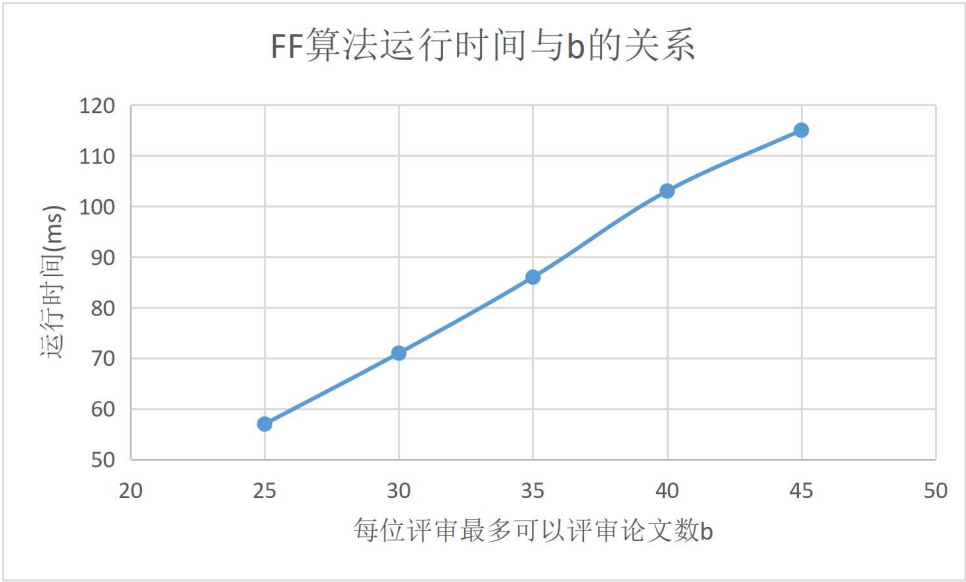
算法正确性得以验证。

4. 性能分析

固定论文数为 500，评审数为 100。

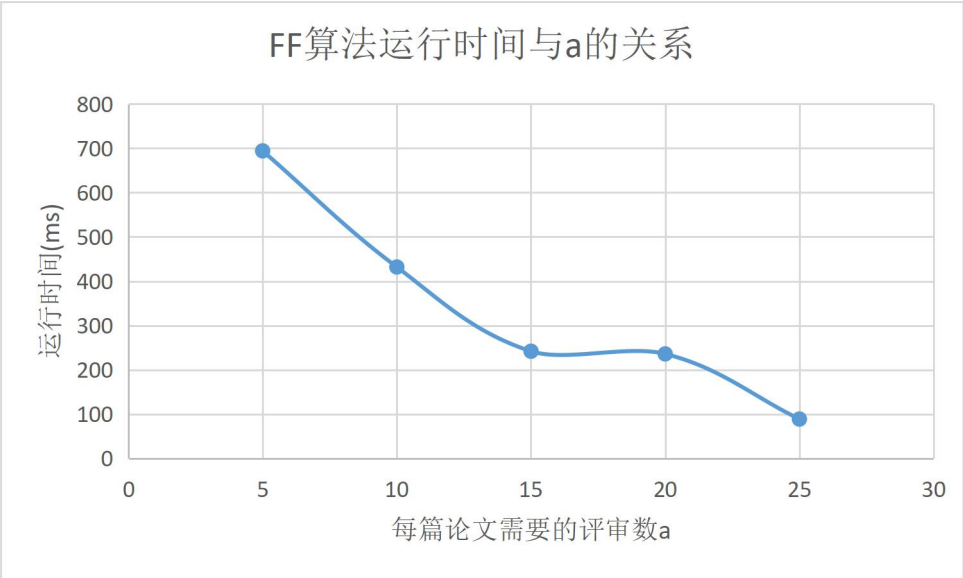
1) 固定 a 的值为 50，增加 b 的值统计运行时间：

每个评审最多可评论文数b	最大流的值	运行时间(ms)
25	2500	57
30	3000	71
35	3500	86
40	4000	103
45	4500	115



2) 固定 b 的值为 30，增加 a 的值统计运行时间：

每篇论文需要的评审数a	最大流的值	运行时间(ms)
5	2500	694
10	3000	432
15	3000	242
20	3000	236
25	3000	89



通过上述对运行时间的统计以及 FF 算法的最坏时间复杂度 O(E|f*|)可以知道，算法实际运行时间与最大流有关，运行时间随最大流值的增大而增大；而最大流保持不变时，运行时间又会随 a 的增大而减小。

（三）Edmons-Karp 算法（以下简称 EK 算法）

1. 解题思路

   该算法的核心思想与 FF 算法一致，只是使用 BFS 代替 DFS 进行增广路径的查找。

2. 伪代码

   根据思路，编写每一个模块功能的伪代码如下：

算法 2 EK

输入: Map 流网络，$start$ 起点，end 汇点

```
1: function EK(start, end)
2:     f ← 0
3:     flow[] ← 0
4:     father[] ← 0
5:     while true do                                          ▷ BFS 寻找增广路径
6:         ANS[start] ← INF                                   ▷ ANS: 残留网络的增广流量
7:         Q.push(start)
8:         while !Q.empty do
9:             u ← Q.front
10:            for v = 1 to end do
11:                if !ANS[v] and Map[u][v] > flow[u][v] then   ▷ u、v 之间有流量且残留量为 0
12:                    ANS[v] ← min(ANS[u], Map[u][v] − flow[u][v])
13:                    father[v] ← u                           ▷ 记录 v 的父亲为 u
14:                end if
15:            end for
16:        end while
17:        if ANS[end] == 0 then                              ▷ 残留网络中不存在增广路径
18:            return f
19:        end if
20:        u ← end
21:        while u!=start do
22:            flow[father[u]][u]+ = ANS[end]
23:            flow[u][father[u]]− = ANS[end]
24:            u ← father[u]
25:        end while
26:        f+ = ANS[end]                                      ▷ 更新最大流
27:    end while
28: end function
```

3. 算法正确性验证

对于 10 篇论文、3 个评委，将 a 赋值为 1，b 赋值为 5。假设评审编号为 A、B、C，论文编号从 0~9，可以任意得到一种分配方案为：A 负责 0~4，B 负责 5~9.（存在多种方案，这里只验证是否有分配方案的判断的正确性）。

输入相应的值，得到结果如下：

```
请输入论文数量m: 10
请输入评审数量n: 3
请输入一篇论文需要安排的评审数a: 1
请输入一个评审最多可评论文数b: 5
The MaxFlow is 10
存在分配方案
the time cost is: 5ms
```

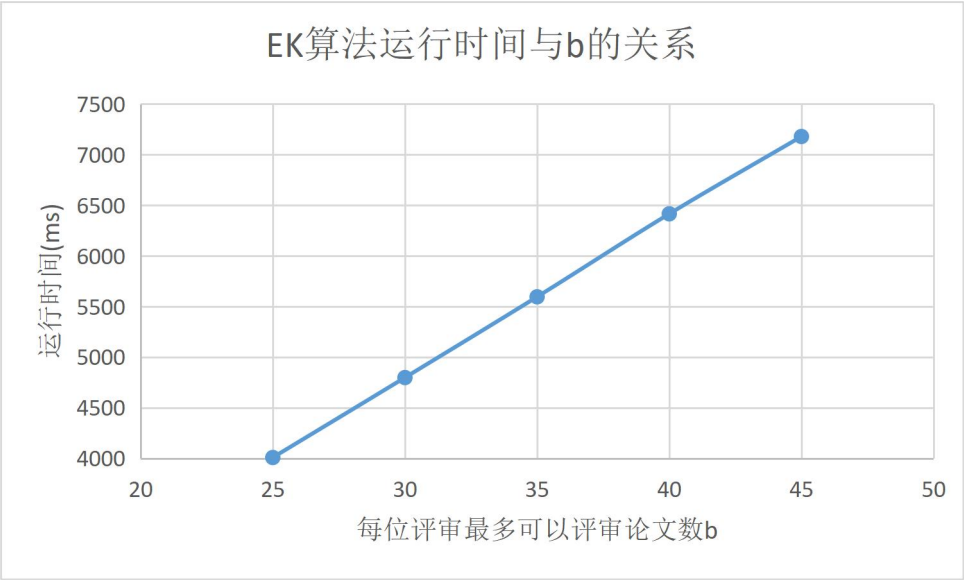
算法正确性得以验证。

4. 性能分析

固定论文数为 500，评审数为 100。

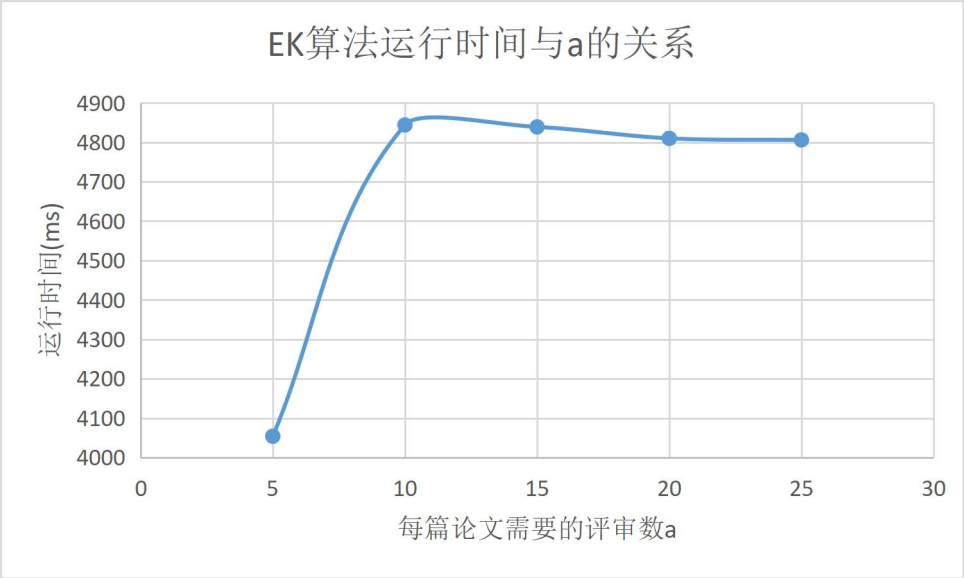
1) 固定 a 的值为 50，增加 b 的值统计运行时间：

每个评审最多可评论文数b	最大流的值	运行时间(ms)
25	2500	4010
30	3000	4799
35	3500	5596
40	4000	6415
45	4500	7178



2) 固定 b 的值为 30，增加 a 的值统计运行时间：

每篇论文需要的评审数a	最大流的值	运行时间(ms)
5	2500	4054
10	3000	4844
15	3000	4839
20	3000	4810
25	3000	4806



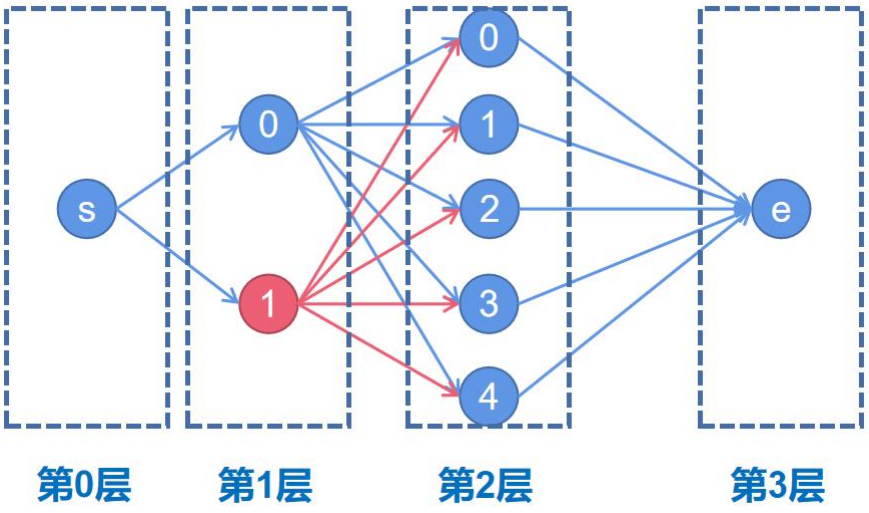
从上述运行时间统计结果可以知道，算法的实际运行时间与 b 的大小有关，运行时间会随 b 的增大而增大；当最大流的值保持不变时，运行时间又会随 a 的增大而减小。

（四）Dinic 算法

1. 解题思路

该算法在上述算法的基础上，进行了分层和多路增广的优化，具体做法如下：

①使用 BFS 分层：采用 BFS 给每个节点标号，标号内容 i 为该点到源点经过的最短边数，根据标号内容 i 把原图构建为一个分层图，每条边均从第 i 层（所在层）连到第 i+1 层；（如下图所示）

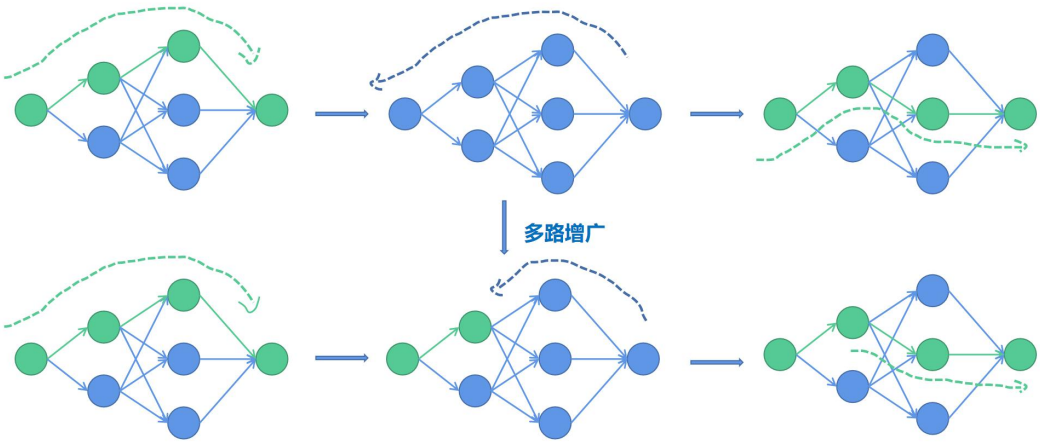


②DFS 寻找增广路径：在分层图上使用 DFS 进行搜索，获得一条从 s 到 e 的路径，获得这条路径上的流量 k（路径上残留最小的边的残量），将增广路径上正向边流量减去 k，反向边流量加上 k（引入多路增广思想，使一次 DFS 可以寻找多条增广路径）；

③重复上述过程，直到不再出现增广路径，获得最大流。

对**多路增广**的具体说明：

我们在使用 DFS 寻找增广路径时，找到一个增广路径以后就直接退回起点重新进行寻找。多路增广思想就是遍历一个增广路径后，只退回到上一个父亲节点，再往后寻找另外的增广路径，实现一次 DFS 寻找多条增广路径的效果。（如下图所示）



2. 伪代码

根据思路，编写每一个模块功能的伪代码如下：

算法 3 Dinic

输入： *Map* 流网络，*start* 起点，*end* 汇点

```
Node:v1, v2 起止点，weight 流量，connect 连接边
layer 层数
function BFS
    layer[] ← 0
    layer[start] ← 1
    Q.push(start)
    while !Q.empty do
        q ← Q.front
        for i = link[q]; i! = 0; i = Edge[i].connect do
            end ← Edge[i].v2
            if layer[end] == 0 and Edge[i].weight! = 0 then
                layer[end] ← layer[q] + 1
                if end == vexnum then                          ▷ 存在增广路径
                    return 1
                end if
            end if
        end for
    end while
end function

function DFS(v, weight)
    if v == vexnum then
        return weight
    end if
    for i = link[v]; i and del; i = Edge[i].connect do
        end ← Edge[i].v2
        if layer[end] == layer[v] + 1 and Edge[i].weight! = 0 then
            key ← DFS(MIN(del, Edge[i].weight))                ▷ weight 为边的剩余流量
            if key == 0 then
                layer[end] ← 0                                 ▷ 不能再流进，作废
            end if
            Edge[i].weight− = key
            Edge[i + 1].weight+ = key
            del− = key
            sum+ = key                                         ▷ sum 记录 v 点具体能够流过的流量
        end if
    end for
end function
```

3. 算法正确性验证

对于 10 篇论文、3 个评委，将 a 赋值为 1，b 赋值为 5。假设评审编号为 A、B、C，论文编号从 0~9，可以任意得到一种分配方案为：A 负责 0~4，B 负责 5~9.（存在多种方案，这里只验证是否有分配方案的判断的正确性）。

输入相应的值，得到结果如下：

```
请输入论文数量m: 10
请输入评审数量n: 3
请输入一篇论文需要安排的评审数a: 1
请输入一个评审最多可评论文数b: 5
The MaxFlow is 10
存在分配方案
the time cost is: 2ms
```

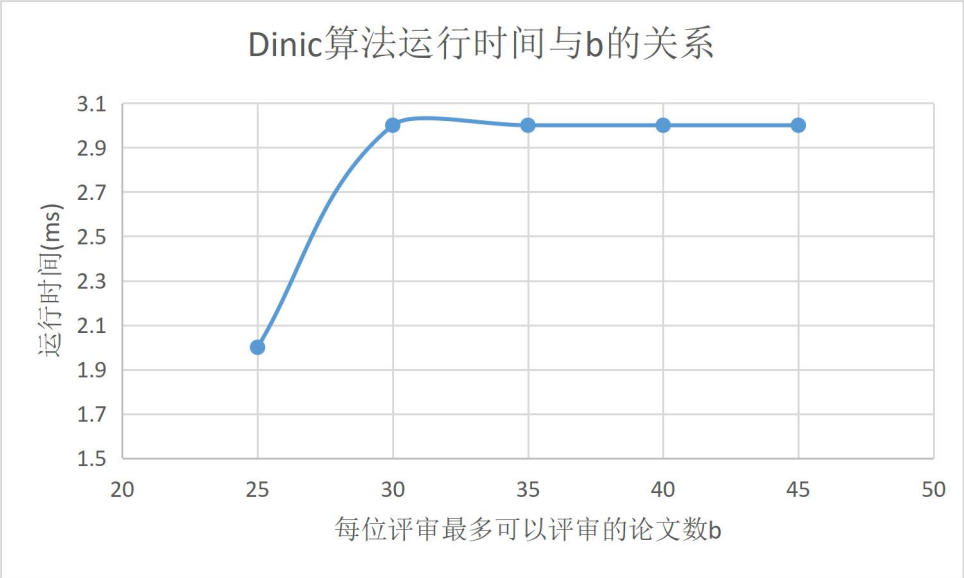
算法正确性得以验证。

4. 性能分析

固定论文数为 500，评审数为 100。

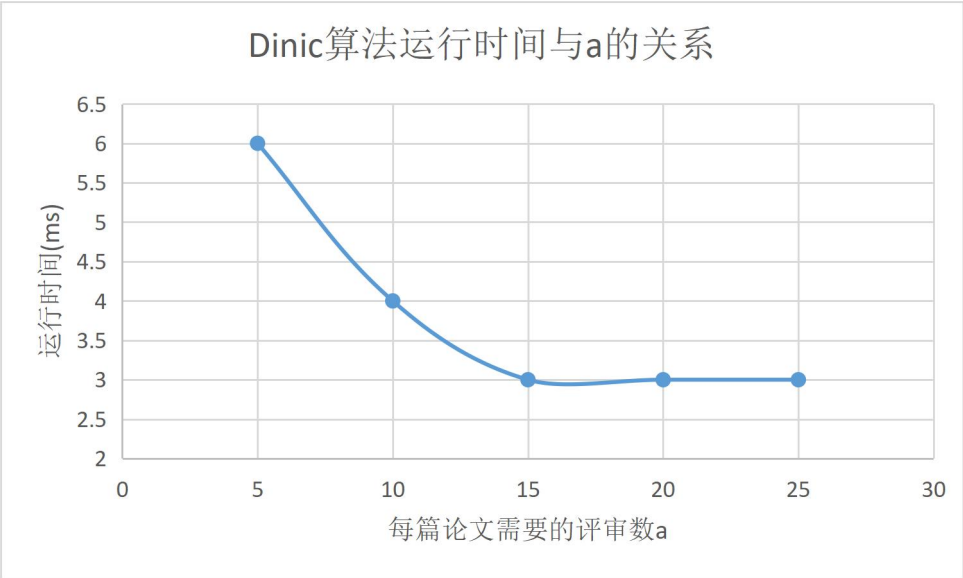
1) 固定 a 的值为 50，增加 b 的值统计运行时间：

每个评审最多可评论文数b	最大流的值	运行时间(ms)
25	2500	2
30	3000	3
35	3500	3
40	4000	3
45	4500	3



2) 固定 b 的值为 30，增加 a 的值统计运行时间：

每篇论文需要的评审数a	最大流的值	运行时间(ms)
5	2500	6
10	3000	4
15	3000	3
20	3000	3
25	3000	3

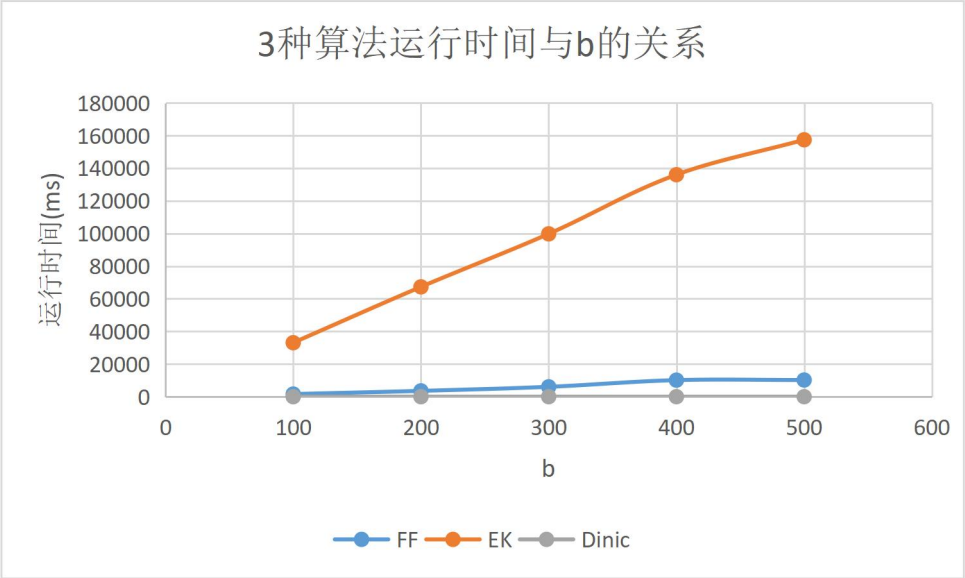


从上述运行时间统计结果可以知道，算法的实际运行时间与 a、b 的大小均有关，运行时间会随 b 的增大而增大，随 a 的增大而减小；当问题不存在分配方案时，运行时间变化趋于平缓。

（五）运行效率分析

1. 固定 m=400，n=200，a=200，增加 b 的值统计 3 种算法的运行时间：

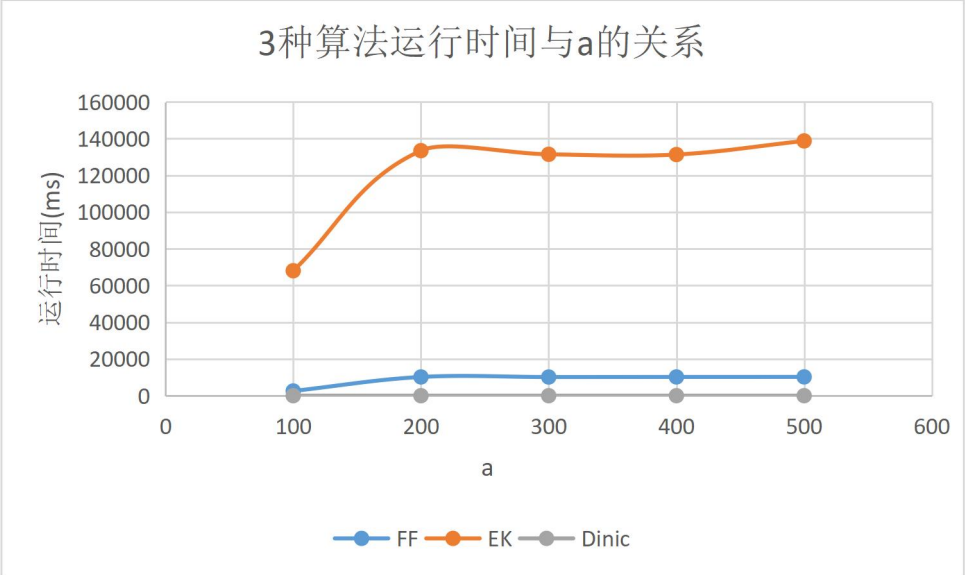
b	FF(ms)	EK(ms)	Dinic(ms)
100	1630	33075	4
200	3581	67297	6
300	6057	99773	6
400	10152	136007	7
500	10224	157392	9



通过上述实验结果可以看到，3 种算法的运行时间都随 b 的增大而增大，其中 EK 算法运行效率最低，运行时间明显高于另外两种算法。

2. 固定 m=400，n=200，b=400，增加 a 的值统计 3 种算法的运行时间：

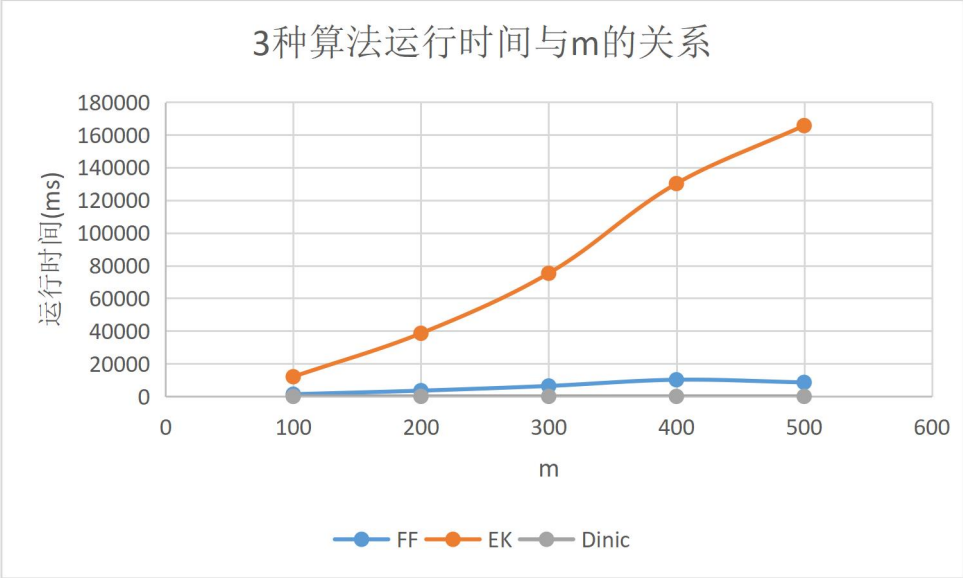
a	FF(ms)	EK(ms)	Dinic(ms)
100	2584	68066	10
200	10208	133445	9
300	10174	131461	7
400	10215	131315	8
500	10242	138748	8



运行时间对 a 的增大会增大，但当分配方案不存在时，运行时间随 a 的变化幅度相比分配方案存在时的变化幅度小得多，基本保持不变。

3. 固定 n=200，a=200，b=400，增加 m 的值统计 3 种算法的运行时间：

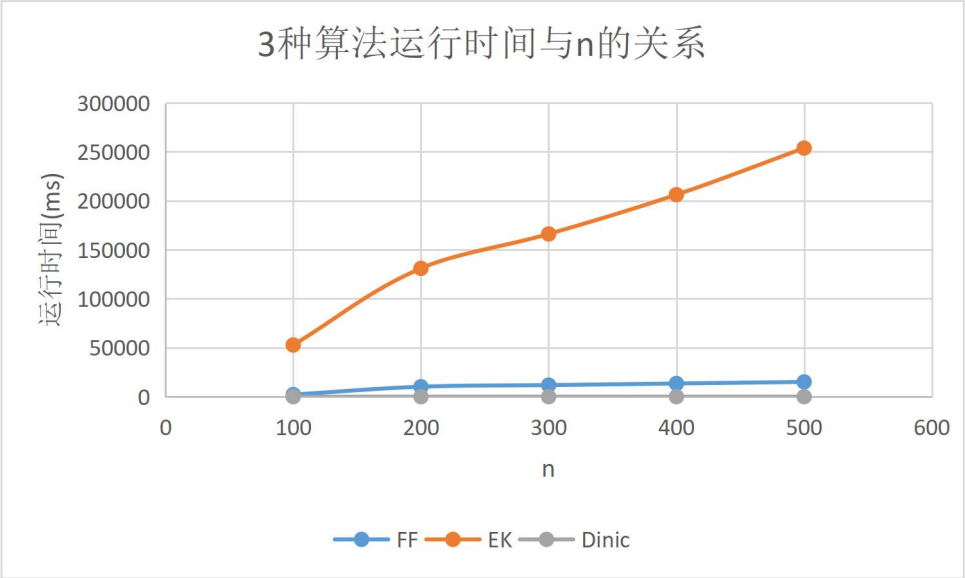
m	FF(ms)	EK(ms)	Dinic(ms)
100	1324	12034	5
200	3512	38581	5
300	6387	75280	7
400	10173	130158	8
500	8562	165611	8



整体上，运行时间随 m 的增大而增大，EK 算法的运行时间最长，且明显高于 FF 算法和 Dinic 算法，Dinic 算法的运行效率最高。

4. 固定 m=400，a=200，b=400，增加 n 的值统计 3 种算法的运行时间：

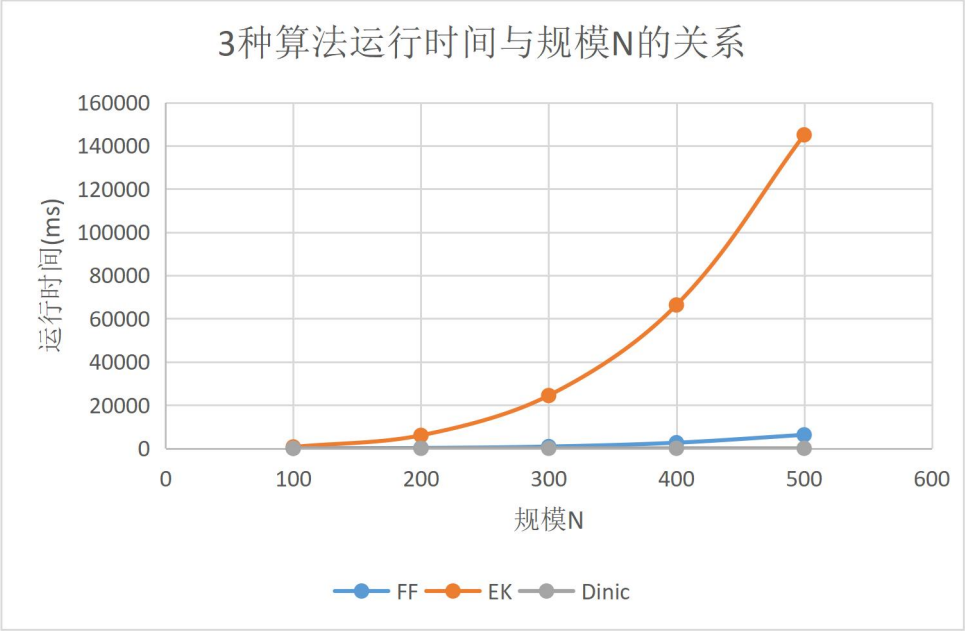
n	FF(ms)	EK(ms)	Dinic(ms)
100	2198	52782	6
200	10200	131060	8
300	11840	166176	16
400	13496	206301	18
500	15103	253848	24



整体运行时间随 n 的增大而增大，算法的运行效率最高是 Dinic，其次是 FF，最低是 EK。

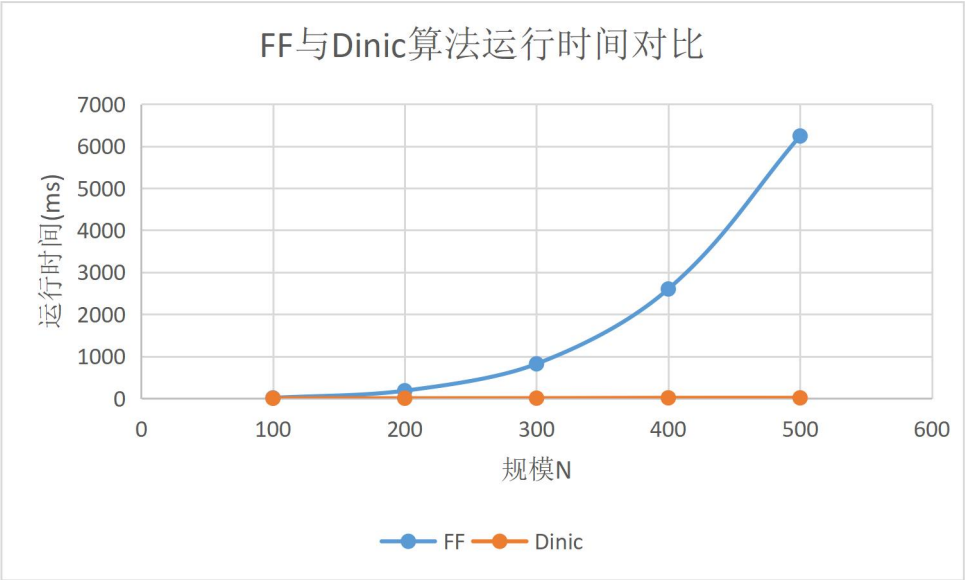
5. 增加规模 N，设定 m=N，n=N/2，a=N/4，b=N，统计 3 种算法的运行时间：

规模N	论文数m	评审数n	a	b	FF(ms)	EK(ms)	Dinic(ms)
100	100	50	25	100	12	643	3
200	200	100	50	200	183	6001	4
300	300	150	75	300	822	24431	6
400	400	200	100	400	2602	66331	13
500	500	250	125	500	6241	145027	13



从上述实验结果可以看到，EK 算法的运行时间远远高于 FF 算法和 Dinic 算法，整体上算法

的运行时间随规模的增大而增大。放大 FF 算法和 Dinic 算法的对比（如下图），可以看到在这两者中，Dinic 的效率也显著高于 FF 算法。可以得出 Dinic 是一种可行的高效算法的结论。



四、实验结论或体会

该实验是利用最大流解决论文评审之间的分配问题。通过该实验学习了如何进行有效的问题转换，同时对 FF 方法下的 EK 算法以及 Dinic 算法进行了学习，通过实验对不同算法的运行效率进行分析，可以知道在实验采用的 3 种算法中，Dinic 算法最高效。我们能也可以借助这一高效算法实现对其他可转换为最大流问题的实际问题进行解决。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。