南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

算法大作业报告

基于供水系统的抽象设计思考：网络流问题

冯思程

年级：2021 级

专业：计算机科学与技术

指导教师：苏明

2023 年 5 月 29 日

摘要

本篇文章从最基本的网络流问题-最大流问题逐步深入网络流的世界进行探索，并根据一些实际情况做出对算法的调整和优化设计，涉及到的算法包括 Ford-Fulkerman 算法、Dinic 算法等等算法，最后对最大流问题进行扩展，分别探究了加入流量下界约束和节点需求的情况下，可行流算法应该如何设计。

关键字：网络流，最大流，供水系统，Ford-Fulkerman 算法，Dinic 算法，可行流算法，算法优化，算法设计

一些基本假设

假设一：容量和流量都是整数而且是非负数

假设二：所有超级源点的供水量为无限

假设二：所有超级汇点的接收供水量为无限

假设三：所有节点的需求和边的容量下界都为整数

目录

一、 研究流程

(一) 第一节-问题提出

在我们日常生活中，供水系统是城市不可或缺的要素。它确保市民的用水需求得到满足，因此供水系统的设计需要经过仔细考虑。一个优秀的供水系统设计需要考虑多个因素，并且可以将其抽象为网络流的问题。因此，我打算从网络流中的最大流问题出发，逐步为系统引入现实条件的约束，并最终设计出一个较为优秀的算法，以满足一些简单的设计需求。

(二) 第二节-改进的最大流问题

第一步对于供水系统的抽象是供水系统的网络流需要有一个源点（供水的水库），多个中间节点（中转供水站），多个汇点（有用水需求的个体户），首先只考虑最大流问题，即，根据给定的数据找出从源点都所有汇点的最大流。

算法设计：由于这里的抽象网络流我考虑到了多个汇点，所以我要根据已有的最大流算法进行一些改进，这里我选取的最大流算法是 Ford-Fulkerman 算法，对于多汇点的处理，我这里想到了一种很好处理办法，即，将所有汇点连到一个汇点（超级汇点），这些边上的流量限制我将其设置成每个个体户承载供水量的上限，这里我将问题转化成了一个基本的最大流问题来进行处理，并利用 Ford-Fulkerman 算法计算，这里避免了对算法内容过程的直接更改，只需要更改输入的时候一些值即可。

Ford-Fulkerman 算法伪代码：

Algorithm 1 Ford-Fulkerman 算法

Input: 抽象图，源点 source，超级汇点 sink
Output: f

```
while 增广图中存在一条从 source-sink 的路径 do
    令 P 是增广图中的一个简单路径
    令 f′ 为 augment(f, P)
    用 f′ 去更新 f
    更新增广图
end while
return f
```

伪代码中的 augment 是一个根据路径产生新流的过程函数，原理是根据新的增广路径进行对前向边和后向边的更新，这个算法十分朴素这里不进行具体说明。另外根据作业报告的要求这里的 c++ 源码将省略，会在电子版中打包提交。

案例构造：在代码中我的输入格式是先输入节点个数（包括源点，中间节点，多个汇点，超级汇点）和边的个数（有向边，包括汇点与超级汇点的相连边，注意：两个节点之间最多有一条边，而且对于多个汇点中的每个，它们有且只有一条出边，出边的终点是超级汇点。）

测试案例如下：（输入格式：第一行是输入抽象图的总节点数和总边数，之后的行是输入边的参数：起点、终点、容量）

```
20 28
0 1 20
0 2 30
0 3 10
1 4 15
1 5 5
2 5 10
2 6 20
3 7 5
3 8 5
4 9 10
4 10 5
5 10 10
5 11 5
6 11 10
6 12 10
7 13 5
8 14 5
9 15 5
10 15 5
10 16 5
11 16 5
11 17 5
12 17 5
13 18 5
14 18 5
15 19 10
16 19 10
17 19 10
```

图 1: 测试案例 1

代码测试结果如下，可以成功找出最大流的流量和:

```
13 18 5
14 18 5
15 19 10
16 19 10
17 19 10
The maximum possible flow is 30
Program ended with exit code: 0
```

图 2: 测试结果 1

(三) 第三节-层次化的供水系统

对于更加现实的供水系统来说，一般是层次化的网络结构，我这里用五级层次进行举例说明(其他更多层次的结构的研究方法相同)，层次更多的问题处理方式是一样的，这里的五层层次是由源点-一级供水站-二级供水站-多个汇点-超级汇点层次结构构成。在其中供水管道的连通规则是在不同级之间的连接相对于同级的连接是稠密的，同级间的连接是稀疏的，这个时候我们可以对最大流算法进行进一步的优化。

对于这个问题，其实我也可以利用最大流的 Ford-Fulkerman 算法直接进行计算，因为对于整个图来说，这些中间节点是否有层次化的结构是无关紧要的问题，我都可以直接对整个图运行 Ford-Fulkerman 算法来进行求解，但是这样求解问题会让求解的复杂度大幅度提高，对此我决定对算法进行优化，适于层次化结构的网络流。

算法设计：这个问题可以视为一种分层图的最大流问题，这与常规的最大流问题有一些不同。由于图是层次化的（即，每一层的节点只能与相邻层的节点连接），这意味着我可以利用这个结构来优化算法。在这种特殊情况下，我决定采用 Dinic 算法。这个算法在 Ford-Fulkerson 的基础上，增加了一种叫做“分层图（layered graph）”或“阻塞流（blocking flow）”的概念。

Dinic 算法首先用 BFS 构造一个分层图，然后在这个分层图上用 DFS 不断寻找增广路径，直到无法找到增广路径为止。这样的一个流称为阻塞流。然后再次进行 BFS 更新分层图，重复以上过程，直到无法再找到增广路径。这样的流就是网络的最大流。

这种方法的优化在于：在每次找增广路径时，都是在 BFS 构造的分层图中进行，这大大减小了搜索空间，从而提高了效率。

下面展示出 Dinic 算法的伪代码：

Algorithm 2 Dinic 算法

Input: 抽象图 G，源点 s，超级汇点 t

Output: f

初始化流为 0
while 图中存在一条从 s-t 的路径 **do**
    用 BFS 构建层级图
    用 DFS 在层级图中寻找增广路径并进行增广
    更新最大流量
end while
return 最大流量

这里需要注意的是 Dinic 算法和 Ford-Fulkerman 算法改进的主要部分就是其中分层图的概念构建，这里算法的主要改进点是优化了路径搜寻，从原来的探索搜寻变成了现在找到最短的增广路径的搜寻模式，针对有类似层次化的网络流结构可以大幅度降低时间复杂度。

案例构造：这里需要注意的是在输入边的时候一定要注意要符合上述的连接规则即可，下面是一个可行的案例：

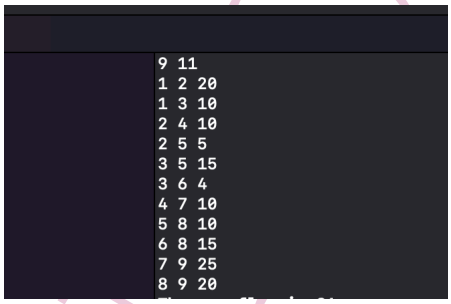

图 3: 测试案例 2

测试结果：

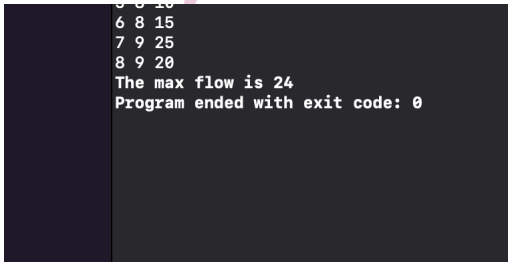

图 4: 测试结果 2

(四) 第四节-在设计中考虑到管道成本

这里我将成本，绑定成与流量成线性相关的函数，即，给出的参数是单位流量下的费用，这也引出了一个网络流中十分著名的问题-最小费用最大流问题。

算法设计：这里的最小费用最大流问题已经有了一些表现较好的算法，由于我已经假设在图中不会出现负数权边，所以根据这一特点，下面我利用 SPFA 算法进行说明（选择这个算法的原因是相对 Bellman-Ford 算法，这个算法普遍更快，而且可以对负权值边进行处理，满足目标要

求)，这里的求解目标就是求一个最大流 f，使流的总输送费用最小。费用是与流量成线性相关的函数。这里 SPFA 本质上就是对 Bellman-Ford 算法进行了队列优化，这里的基本算法逻辑就是每次都去寻找最短路径的增广路径，直到找不到路径了为止，这里的最短就是费用最小，费用在这里是与流量成线性相关的函数，每条边的系数都在输入中给出。

下面展示出 SPFA 算法的伪代码：

Algorithm 3 SPFA 算法

Input: 网络图 G，源点 s，汇点 t
Output: 最小费用流 minCost 和最大流量 maxFlow

初始化 minCost 和 maxFlow 为 0
while 存在从 s 到 t 的增广路径 P **do**
    使用 SPFA 寻找从 s 到 t 的最小费用增广路径 P
    根据路径 P，找出该路径上的最小残量 f
    将路径 P 上的每条边的流量增加 f，反向边流量减少 f
    更新最大流量 maxFlow += f，更新最小费用 minCost += f * cost[P]
end whilereturn 最小费用流 minCost 和最大流量 maxFlow

案例构造：这里需要注意的是输入格式要与代码的输入格式相匹配，下面是一个可行的案例：

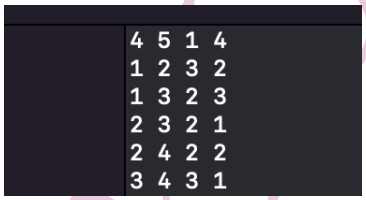

图 5: 测试案例 3

测试结果：

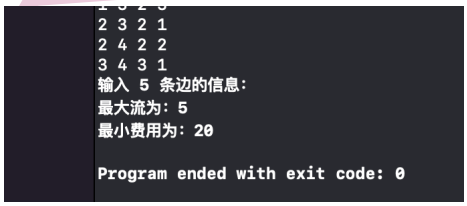

图 6: 测试结果 3

(五) 第五节-最大流问题的扩展

现在考虑多源多汇的情况，从现在不考虑最大流的求解，而是考虑源点有固定供给值，汇点有固定需求值，现在求解的目标转化成了利用有效的供给来满足所有需求，现在对图中的每个节点给定一个 d_v 作为其需求，如果 d_v 大于 0，则这个点是汇点，它希望 input 的流要比 output 的流多出 d_v，如果 d_v 小于 0，则这个点是源点，它希望 input 的流要比 output 的流少 d_v。如果 d_v 等于 0，则这个点是中间节点，它希望 input 的流和 output 的流相等。这里对于源点和汇点没有对出入边存在性的限制，源点集合我命名为 S，汇点集合我命名为 T，这里 S 中的节点尽管发送的流要大于接收的流，它也允许 S 的节点可以有入边上有流流入，同理对于汇点也成立(相反的方向)，下面对容量和需求条件进行形式化展示：

1. 对于每个 $e \in E$，我们有 $0 \leq f(e) \leq c_e$

2. 对于每个 $v \in V$，我们有 $f_{in}(v) - f_{out}(v) = d_v$

现在我们考虑是否存在一个设计使得可以满足上述两个条件，首先易知为了能够寻找到符合要求的设计，这里有一个前提就是总供给应该等于总需求，也就是 d_v 求和为 0。

算法设计：这里应用到的方法是将寻找可行的设计问题等价转换到在另一个网络中最大流问题，而网络的变换方式有点类似于在处理多汇点问题时候的处理方式，即，我添加一个超级源点，并为 S 中的每个节点赋予一条边，边的容量是-d_v，边的端点是（s_*，u）；再添加一个超级汇点，为 T 中的每个节点赋予一条边，边的容量是 d_v，边的端点是（v，t_*），其余网络结构保持不变。在这个新图中如果我们能找到一个值为 $\sum_{v \in S}(d_v)$ 的最大流就可以说明在原图中存在一个可行流通，而且这里根据假设，会存在一个整数值的可行流通。可视化展示如下：

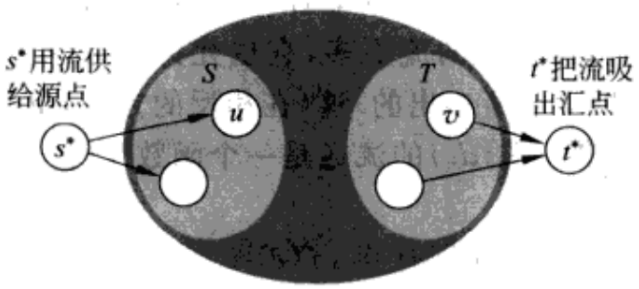

图 7: 可行流通问题归为最大流问题的网络结构变化

让我根据一个例子来进行解释，下图中节点内的数字是需求，边上标记的数是容量和流量，流量数值用方框圈住。在（a）图中两个节点是源点，需求都是-3，另两个节点是汇点，需求是 2 和 4，图中的流量组成了一个可行流通，那么可以把这个实例归为在（b）图中的等价最大流问题的结果，在（b）图中展示了流量达到最大流的一个流，与（a）图相互印证。

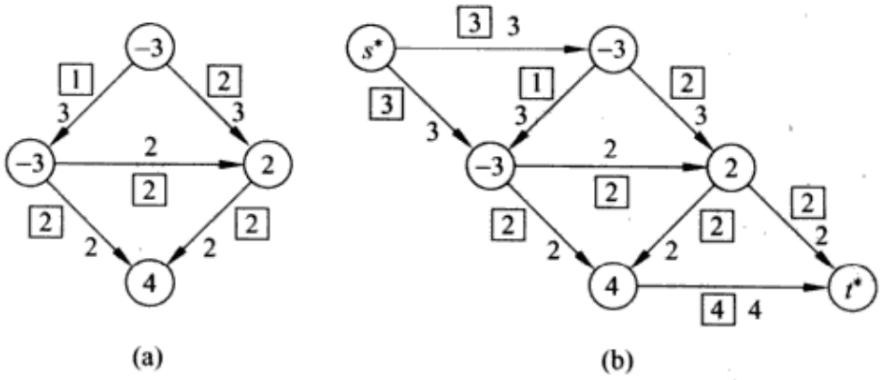

图 8: 最大流扩展需求例子对比图

现在让我们更加一步，我希望对边的限制再加上一个下界，这也与现实的供水系统设计息息相关，为了保证管道的充分利用，每个管道会有一个最小流量确保这个管道的利用效率不会过低。现在我的流通条件变为了，如下：

1. 对于每个 $e \in E$，我们有 $l_e \leq f(e) \leq c_e$

2. 对于每个 $v \in V$，我们有 $f_{in}(v) - f_{out}(v) = d_v$

现在我的目标和之前相同还是寻找一个可行的流通。

算法设计：这里我的策略是将其转化为带节点需求，但是没有对边的下届约束的可行流通问题，因为如果我可以成功转化问题，那么问题就可以转化为一个普通的最大流问题，可以轻易解决。思想类似上文，我同样想利用一个网络的转换来实现对问题的转化，首先我们知道对于每条边流量至少要达到 l_e，所以我先通过 $f_0(e) = l_e$ 定义一个初始的流 f_0，可以知道这个流满足了所有的流量下界约束和容量约束，但是不一定满足需求要求。此时对于每个节点有下面等式：

$$L_v = \sum_{e->v} (l_e) - \sum_{v->e} (l_e) \tag{1}$$

如果 L_v 等于 d_v 则说明该点的需求条件被满足，如果不相等，我期望利用某种方法来对其进行调整，使其可以满足其需求条件，这里在 f_0 的基础上再加入一个流 f_1，它的目标是去除掉节点上剩余的这种“不相等”对于不相等的节点，我需要的流量是 $d_v - L_v$，而我有多少容量可供我进行调整呢，对于每条边有 $c_e - L_e$ 容量供我使用。

根据上述的说明可以自然的想到下面的图结构：边和节点的结构不变化，边的下界约束不考虑，边的容量变为 $c_e - L_e$，节点的需求更新为 $d_v - L_v$，现在就成功转坏为带节点需求，但是没有对边的下届约束的可行流通问题了，接下来的处理同理上文，最后可以转化为一个普通的最大流问题，即可成功解决问题，另外这里根据假设，可以得出如果存在可行流通即可推出存在整数值的可行流通。

补充：这里的最大流算法我利用的是 Dinic 算法，在案例表现上速度较快。

算法的伪代码：

Algorithm 4 可行流算法

Input: 网络图 G，需求 Demand，容量 Capacity，下界 LowerBound

Output: 是否存在可行流

```
创建两个数组 'ExcessDemand'和 'ResidualCapacity'，分别用于存储每个节点的需求差额和每条边的剩余容量
for 每一条边 e = (u, v) do
    更新需求差额：'ExcessDemand[u] += LowerBound[u][v]' 和 'ExcessDemand[v] -= LowerBound[u][v]'
    更新剩余容量：'ResidualCapacity[u][v] = Capacity[u][v] - LowerBound[u][v]'
end for
创建超级源节点 s* 和超级汇点 t*
for 每一个节点 v do
    if 'ExcessDemand[v] < 0' then
        添加一条从 s* 到 v 的边，其容量为 '-ExcessDemand[v]'
    else if 'ExcessDemand[v] > 0' then
        添加一条从 v 到 t* 的边，其容量为 'ExcessDemand[v]'
    end if
end for
使用最大流算法在超级源节点和超级汇点之间找出最大流
if 最大流等于所有连接到超级源的边的容量总和 then
    return 'Yes'
else
    return 'No'
```

19: **end if**

案例构造：

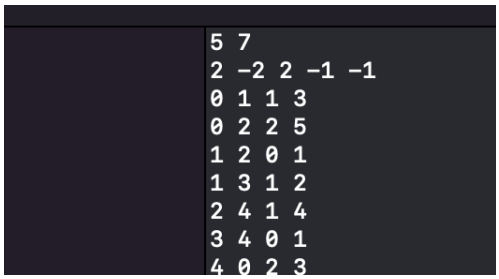

图 9: 测试案例 4

测试结果：

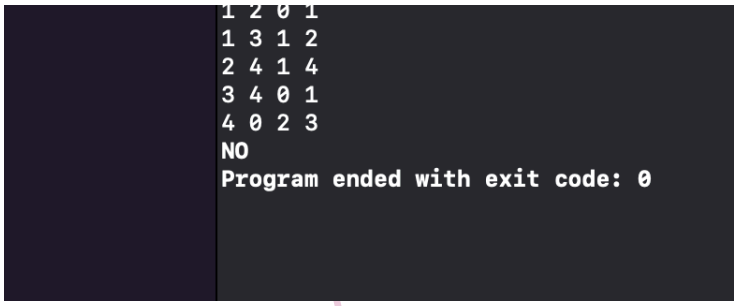

图 10: 测试结果 4

二、 总结

(一) 第一节-关于上述涉及到的代码的时间复杂度和空间复杂度分析

列表进行对比展示:（E 代表边数，V 代表节点数）

复杂度（O 表示模式）\ 算法	Ford-Fulkerman 算法	Dinic 算法	SPFA 算法	可行流算法
时间复杂度	V^2E	$(V+E)F$	$(V+E)F$	V^2E
空间复杂度	V^2	V+E	V+E	V+E

表 1: 时间空间复杂度对比表

(二) 第二节-上述算法的内在联系

他们都是解决网络流领域中的问题，其中 Ford-Fulkerman 算法是最基础的最大流问题算法，Dinic 算法是对 Ford-Fulkerman 算法进行优化的算法，然后针对最小费用最大流问题，SPFA 算法是一个可以处理负权值速度较快的算法，其中也包含了最大流的求解过程，SPFA 算法是对 Bellman-Ford 算法进行了队列优化而得来的，最后我加入更加符合现实的元素-容量下界和节点需求，在最大流的基础上提出了可行流算法。

三、 不足与未来发展方向

(一) 第一节-一些不足之处

在本篇报告中，我只是较为浅略的说明了在网络流问题中的一些算法，在深入理解算法的原理和细节方面，我可能还做的不够好，还应该深入理解其具体实现细节、关键数据结构和优化技巧。

在对于算法的适用性和场景选择方面，我只是进行了简单的说明和应用，并没有仔细去深入挖掘。不同的算法在不同的问题和数据规模下可能表现出不同的效果。这一点也是我做的还可以进步的部分。

算法的实际应用和问题建模：学习算法不仅仅是理解其原理和实现，还需要将其应用于实际问题。这包括将问题转化为适合算法求解的形式，设计合适的数据结构和算法实现，并对算法的输出结果进行正确性验证。

对于算法的复杂度分析和优化方面，我没有针对每个算法进行充分的优化，例如 SPFA 算法所应用的最小费用最大流问题上还有改进的 Dijkstra 算法可以进一步在某些场景提高速度。

对于可行流问题，在上文中探究的依然是一个较为简单的算法，我还应该去更多的了解其他与可行流相关的算法和扩展，如最小割、多源最短路径等。

(二) 第二节-未来发展方向

在本篇报告中的网络流的流量是静态的，那么如果变成动态会是什么样子呢，在现实世界中，网络流也是动态变化的，例如交通网络等等，对于动态网络流的研究可能会是一个更加火热的方向。

对于多目标网络流问题，在实际应用中，我们往往需要同时考虑多个目标，例如最大化流量、最小化费用、最小化延迟等。然而，这些目标之间可能会是相互冲突的，需要进行加权平衡等操作。这个问题也可能是一个重要的研究方向。

对于网络流在现实中的应用研究还有很多可以进行深入研究，网络流的理论已经相对成熟，然而在特定领域的应用上还有很多空白。例如在现代供应链、物流、电力网络等复杂系统中，网络流的应用还有很大的研究空间。

对于网络流算法的优化方面，有的传统网络流算法在大规模问题上可能会遇到效率问题。因此，如何提高算法的时间复杂度和空间复杂度可能是一个重要的研究方向，例如通过并行化的计算等等手段。

四、 参考资料

1. 算法设计，张立昂，屈婉玲译版

2. CSDN 博客，链接：点击这里访问目标地址

3.《计算机应用》2008 年第一期，点和边有容量约束的网络最大流新算法，厍向阳、罗晓霞，链接：点击这里访问目标地址