APPLICATIONS OF LINEAR ALGEBRA IN NETWORK FLOW

Hanzhe Huang, Xicheng Li, Xinzhu Wang, Yannuo Wen

Beijing-Dublin International College
Beijing University of Technology[1]

Abstract

Network flow models were widely used in the research of physical network such as transportation networks, electrical distribution systems and the food web in the ecology system. Nowadays, people are trying to apply computer science algorithms in solving network flow problems. Because of the rapid increment of data, we are facing the bottlenecks of machine performance and algorithms efficiency, so that in this paper we made a difference that using linear algebra to solve the network flow problems instead of common algorithms and apply it to computer science algorithms.

Keywords: Linear Algebra, Network Flow, computer science algorithms, Time Complexity

Nomenclature

$C_{i,j}$ The weight of the edge which from node i to node j

E A set of V's edges.

$f(u, v)$ Flow from u to v.

$G = (V, E)$ A graphic with V and E.

O Big O notation, which used to describes the limiting behavior of a function.

S The source.

T The target.

V A set of vertices.

BFS Breadth first search algorithm.

DFS Depth first search algorithm.

[1]No.100, Pingleyuan, Chaoyang Distric, Beijing, China.

1. Introduction

The algorithms of network flow enable the network flow system becomes better projected and more rationally used, and provided a convenience for manufacture and livelihood of people, such as the allocation of electrical power system, the project for logistics system, the circuit design and so on, hence designing an optimized network system marks the ultimate goal for many engineers. Nowadays, various algorithms were developed for solving a series of network flow problems by many science researchers in computer and mathematics realms. For example, the Dinic's algorithm, put forward by a former Soviet Union scientist Yefim(Chaim)A. Dinitz[1][2], is a very classical algorithm in solving the maximum network flow problem.

Furthermore, algorithms like Ford-Fulkerson and Edmonds-Karp are also able to settle this issue under different time complexity[3][4]. Though we already have various ways to solve the set of network flow problems, the realization of all these traditional algorithms need redundant traverse of nodes in the whole network.

To minitype networks, the expenditure of time brought by the complexity of the traverse can be ignored, but in a large-scale network system, such as the distribution of electronic power system and the projection of logistics network the matter must be put on the stage. The process of large-scale linear equations used an equivalent method of coefficient matrix, such that the amount of calculation and the algorithm get simplified and unified.Currently speaking, literature about the application of linear algebra in network flow system is extremely insufficient. So our study tries to apply the thoughts which use linear equation quantization and matrix to solve problems to the problem of the maximum network flow.

In this paper, we disposed every side as an upper bond limited unknown variable. Expounding the Kirchhoff's current law and following the rule that the influx of every network node equals to the outflow volume, we created a set of linear equations and system of inequalities with a bigger amount than the unknown variables, and turned them into linear programming system. Finally, we used simplex method to get optimized solution, i.e., to get the maximum flow. To decrease the trouble of manual calculation, we realized the progress by C++ and carried out estimation.

2. Materials and Methods

2.1. Test Data Generation

It will cost us a lot time to start writing to write a random graph theory data generator program from nothing into something, so that we decided to choose CYaRon[5], a open source software which was used to generate input data for olympic-informatics problems. By using CYaRon, we could generate graph data information with edges and nodes conveniently and get the output data of different programs immediately.

2.2. Mature Algorithms

2.2.1. Implement of Dinic's Algorithm and Ford-Fulkerson Algorithm

The original version of Ford-Fulkerson[3] algorithm is finding a route from S to T constantly and building the residual network, then trying to find another route on the residual

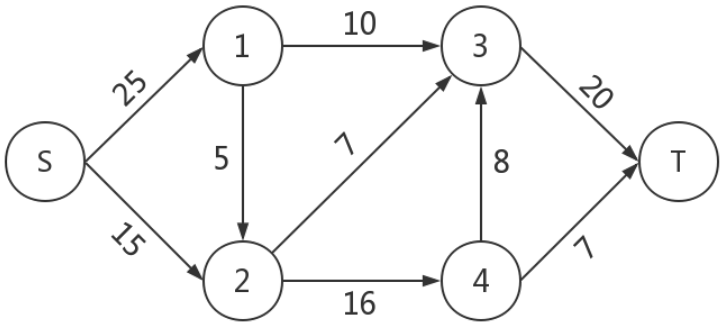
network and looping until there is no other available route on the residual network. However, it will cost us a lot of time to do the BFS in each turn of augmentation. To optimize this, Dinic's algorithm tries to find numbers of augmenting routes in each turn to reduce the time cost.

Firstly, Dinic's algorithm delaminates the residual network with BFS by the minimal number of edges of the route from S to the node. Next, use DFS to find the augmenting routes layer by layer and do the procedure above in loop until there is no available route from S to T.

2.2.2. Implement of Edmonds-Karp Algorithm

Edmonds-Karp algorithm, an another computer science algorithm, which is based on the Ford-Fulkerson algorithm, runs in $O(VE^2)$ time, using DFS instead of BFS to find the route in loop.

2.3. Linear Algebra Application in Algorithm



(1)

Here is a sample figure(1), which consisted of 6 nodes and 9 edges. To quantify it to a mathematics model in order to calculate, we set every edge as an upper bond limited non-negative variable. (For a convenient observation, we use two nodes corresponding to an edge to express the variable, for an example, we can denote the edge between S and node 1 by $C_{S,1}$, then $c_{S,1} \leqslant 25$.

After setting up the variables, we get the linear in-equations under the condition of the influx equals to the outflow. Analyzing the node 1 we can get $C_{S,1} = C_{1,2} + C_{1,3}$. Finally, we can get linear equations and the augmented matrix corresponding as follows(2):

$$\begin{cases} C_{S,1} + C_{S,2} = C_{3,T} + C_{4,T} \\ C_{S,1} = C_{1,3} + C_{1,2} \\ C_{S,2} + C_{1,2} = C_{2,3} + C_{2,4} \\ C_{1,3} + C_{2,3} + C_{4,3} = C_{3,T} \\ C_{2,4} = C_{4,3} + C_{4,T} \end{cases} \Leftrightarrow \left[\begin{array}{ccccccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 \end{array}\right] \tag{2}$$

Meanwhile, the scope of every non-negative variable can be given by linear in-equations, and we can also solve its corresponding augmented matrix(3).

$$\begin{cases} C_{S,1} \leqslant 25 \\ C_{S,2} \leqslant 15 \\ C_{1,2} \leqslant 5 \\ C_{1,3} \leqslant 10 \\ C_{2,3} \leqslant 7 \\ C_{2,4} \leqslant 16 \\ C_{4,3} \leqslant 8 \\ C_{3,T} \leqslant 20 \\ C_{4,T} \leqslant 7 \end{cases} \Leftrightarrow \left[\begin{array}{c|c} C_{S,1} & 25 \\ C_{S,2} & 15 \\ C_{1,2} & 5 \\ C_{1,3} & 10 \\ C_{2,3} & 7 \\ C_{2,4} & 16 \\ C_{4,3} & 8 \\ C_{3,T} & 20 \\ C_{4,T} & 7 \end{array}\right] \tag{3}$$

Thus, the maximum network flow problem is transformed into solving $maxz_1 = C_{S,1} + C_{S,2}$ or $maxz_1 = C_{3,T} + C_{4,T}$.

Then, the next purpose is to solve the model consisted of the set of equations and in-equations. Here we use indiscriminately a general solution of linear programming: simplex method.[6]

$$\begin{cases} A_1 = 25 - C_{S,1} \geqslant 0 \\ A_2 = 15 - C_{S,2} \geqslant 0 \\ A_3 = 5 - C_{1,2} \geqslant 0 \\ A_4 = 10 - C_{1,3} \geqslant 0 \\ A_5 = 7 - C_{2,3} \geqslant 0 \\ A_6 = 16 - C_{2,4} \geqslant 0 \\ A_7 = 8 - C_{4,3} \geqslant 0 \\ A_8 = 20 - C_{3,T} \geqslant 0 \\ A_9 = 7 - C_{4,T} \geqslant 0 \end{cases} \tag{4}$$

First we use the method of substitution to transform the limiting condition of variables into a format of $A_i \geqslant 0$(4).

Then use the variable Ai to transform the equation system into standard linear programming system(5)(6)(7).

$$\begin{aligned} maxz_1 &= 20 - A_8 + 7 - A_9 \\ &= 25 - A_1 + 15 - A_2 \end{aligned} \tag{5}$$

$$\begin{cases} 25 - A_1 + 15 - A_2 = 20 - A_8 + 7 - A_9 \\ 25 - A_1 = 10 - A_4 + 5 - A_3 \\ 15 - A_2 + 5 - A_3 = 7 - A_5 + 16 - A_6 \\ 10 - A_4 + 7 - A_5 + 8 - A_7 = 20 - A_8 \\ 16 - A_6 = 8 - A_7 + 7 - A_9 \end{cases} \tag{6}$$

$$A_1, A_2, \ldots, A_9 \geqslant 0 \tag{7}$$

Then using the solution of simplex method, and by finding the maximum value of target function Z2 using iteration, we can solve the maximum flow.

3. Results and Conclusion

Our essay successfully planted the thought of solving linear programming into the maximum network flow problem (such as using linear equations to solve the flow of nodes, applying transformation of matrices with respect to linear algebra knowledge system in simplex method), therefore realized the appliance of linear algebra in network flow problems.

Comparing with solution of other fashioned algorithms, this kind of method possess a lower difficulty of understanding and that of operating, which can be applied in both minitype network calculation and solving complex large-scale network flow problem after realized by codes.

Appendix A. Algorithms Source Codes

Appendix A.1. Edmonds-Karp Algorithm Template in C++

```
/**
 * Author: myzonehi
 * Date   : 2011/05/26
 * From   : http://blog.sina.com.cn/s/blog_6cf509db0100uy5n.html
 */
#include<iostream>
#include<queue>
#include<memory.h>
using namespace std;
const int maxn = 205;
const int inf = 0x7fffffff;

bool visit[maxn];
int r[maxn][maxn];
int pre[maxn];
int m, n;

bool bfs(int s, int t)
{
    int p;
    queue<int> q;
    memset(pre, -1, sizeof(pre));
    memset(visit, false, sizeof(visit));
    pre[s] = s;
    visit[s] = true;
    q.push(s);
    while(!q.empty())
    {
        p = q.front();
        q.pop();
        for(int i = 1; i <= n; i++)
        {
            if(r[p][i] > 0 && !visit[i])
            {
                pre[i] = p;
                visit[i] = true;
                if(i == t)
                    return true;
                q.push(i);
            }
        }
    }
    return false;
}

int EdmondsKarp(int s, int t)
{
    int flow = 0,d,i;
```

```
    while(bfs(s, t))
    {
        d = inf;
        for(i = t; i != s; i = pre[i])
            d = d < r[pre[i]][i] ? d : r[pre[i]][i];
        for(i = t; i != s; i = pre[i])
        {
            r[pre[i]][i] -= d;
            r[i][pre[i]] += d;
        }
        flow += d;
    }
    return flow;
}


int main()
{
    while(scanf("%d%d", &n, &m) != EOF)
    {
        int u,v,w;
        memset(r, 0, sizeof(r));///
        for(int i = 0; i < m; i++)
        {
            scanf("%d%d%d", &u, &v, &w);
            r[u][v] += w;
        }
        printf("%d\n", EdmondsKarp(1, n));
    }
    return 0;
}
```

Edmonds-Karp-Algorithm.cpp[7]

Appendix A.2. Dinic Algorithm Template in C++

```
/**
 * Author : Comzyh
 * From   : https://comzyh.com/blog/archives/568/
 * Note   : ONLY COULD RUN UNDER 250 * 250
*/
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#define min(x, y) ((x < y) ? (x):(y))
using namespace std;
const int MAX = 0x5fffffff;//
int tab[250][250];
int dis[250];
    int q[2000], h, r;
```

```
int N, M, ANS;
int BFS()
{
    int i, j;
    memset(dis, 0xff, sizeof(dis));
    dis[1] = 0;
    h = 0;
    r = 1;
    q[1] = 1;
    while (h < r)
    {
        j = q[++h];
        for (i = 1; i <= N; i++)
        if (dis[i] < 0 && tab[j][i] > 0)
        {
            dis[i] = dis[j] + 1;
        q[++r] = i;
        }
    }
    if (dis[N] > 0)
        return 1;
    else
        return 0;
}
int find(int x, int low)
{
    int i, a = 0;
    if (x == N)
        return low;
    for (i = 1; i <= N; i++)
        if (tab[x][i] >0 && dis[i]==dis[x]+1 && (a=find(i,min(low,tab[x][i]))
            ))
        {
            tab[x][i]-=a;
            tab[i][x]+=a;
            return a;
        }
    return 0;
}
int main()
{
    freopen("test.in","r",stdin);
    freopen("test.out","w",stdout);
    int i, j, f, t, flow, tans;
    while (scanf("%d%d", &N, &M) != EOF)
    {
        memset(tab, 0, sizeof(tab));
        for (i = 1; i <= M; i++)
        {
                scanf("%d%d%d", &f, &t, &flow);
                tab[f][t] += flow;
```

```
        }
        ANS = 0;
        while (BFS())
        {
            while(tans = find(1,0x7fffffff))
                ANS+=tans;
        }
        printf("%d\n",ANS);
    }
    fclose(stdin);
    fclose(stdout);
}
```

Dinic-Algorithm.cpp[8]

Appendix B. CYaRon Config

```
from cyaron import *

for i in range(0, 10):
    test_data = IO(file_prefix = "test", data_id = i)
    node_num = 50
    edge_num = 100
    graph = Graph.graph(node_num, edge_num, directed = True, weight_limit =
        (1, 100), repeated_edges = False, self_loop = False)
    for edge in graph.iterate_edges():
        edge.start
        edge.end
        edge.weight
        test_data.input_writeln(edge)
```

Generate.py

References

[1] Y. C. A. Dinitz, Algorithm for solution of a problem of maximum flow in a network with power estimation, Doklady Akademii nauk SSSR 11 (5) (1970) 1277–1280.

[2] Y. C. A. Dinitz, Dinitz's algorithm: The original version and even's version[Online; Accessed on 7th December, 2017].
URL https://www.cs.bgu.ac.il/~dinitz/Papers/Dinitz_alg.pdf

[3] D. R. Fulkerson, L. R. Ford, Maximal flow through a network[Online; Accessed on 7th December, 2017].
URL http://www.cs.yale.edu/homes/lans/readings/routing/ford-max_flow-1956.pdf

[4] J. R. Edmonds, R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Association for Computing Machinery 19 (2) (1970) 248–264. doi:10.1145/321694.321699.

[5] L. D. Team, Cyaron: Yet another random olympic-informatics test data generator[Online; Accessed on 7th December, 2017].
URL https://github.com/luogu-dev/cyaron

[6] muzhi, Simplex algorithm to solve linear programming problems in c++[Online; Accessed on 7th December, 2017].
URL http://blog.csdn.net/zhoubin1992/article/details/46916429

[7] myzonehi, Maximal flow template - edmonds-karp algorithm[Online; Accessed on 7th December, 2017].
URL http://blog.sina.com.cn/s/blog_6cf509db0100uy5n.html

[8] Comzyh, Maximal flow introduction - dinic algorithm used in network flow[Online; Accessed on 7th December, 2017].
URL https://comzyh.com/blog/archives/568/