Modelling District Heating and Cooling network

Leanne Dong

Gina Cody School of Engineering and Computer Science
Concordia University Montréal

July 1, 2020

Overview

District Heating and Cooling Networks (DHN)

- The most common method for heating building in cities
- Usually consist of supply and return pipes that deliver heat in form of hot water or stream

Common graph problems

- Shortest Path Problem : Breadth First Search (BFS), Dijkstra's, Bellman-Ford, Floyd-Warshall
- Connectivity/Path finding: A* algorithm, BFS, DFS
- Negative Cycle : Bellman-Ford, Floyd-Warshall
- Traveling Salesman Problem:
- Bridges: These are edges in a graph whose removal could increase the number of connected components in the graph. They are important because they represent the vulnerabilities and bottlenecks with in the graph.
- Minimum Spanning Tree: Kruskal's and Prim's algorithm
- Maximum Network Flow: Ford-Fulkerson and Edmonds Karp& Dinic's algorithms

Our graph problems

Problem 1: : Suppose there are 1000 building in a town, can we implement an algorithm that connects all the thousand building? Implement a A* algorithm.

The **Depth First Search (DFS)** is the most fundamental search algorithm used to explore nodes and edges of a graph. It runs with a time complexity of $O(V + E)$ and is often used as a building block in other algorithms.
By itself the DFS isn't all that useful, but when augmented to perform other tasks such as

- Count connected components,
- Determine connectivity,
- Find bridges/articulation point.

then DFS really shines.

General problems one may ask first

Ask yourself:

- Is the graph directed or undirected?
- Are the edges of the graph weighted?
- Is the graph we will counter likely to be sparse or dense with edges?
- Should I use an adjacency matrix (or incidence matrix), adjacency list, an edge list or other structure to represent the graph efficiently?

Network graph

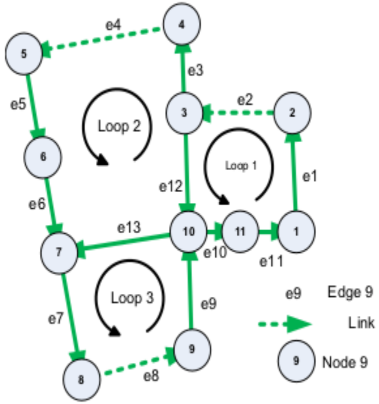

Figure: Network graph of the DH network in Scharnhauser Park

Assumption

- The basic assumption for the calculation is incompressible media.
- Computation of flow distributions in DHN are mainly based on the Kirchhoff law for current and voltage in circuits: The two equations describe flow rate and pressure losses in the network
- Consider the PDE describing an one dimensional flow through a horizontal pipe which can be systematically derived from the Navier-Stokes equations.

$$\frac{l}{A}\frac{d\dot{m}}{dt} + \Delta p + R|\dot{m}|\dot{m} = 0 \tag{1}$$

Note: Δp denoting the difference in pressure head between the two pipes ends and $\dot{m}$ is the mass flow rate. The variable R stands for the hydraulic resistance of the pipe element, which is postulated to be a function of the physical properties such as length, roughness and diameter.

Analogy of rules

Electrical network	Kirchoff's current law	Kirchoff's voltage law	Ohm's Law
District heating network	Continuity of flow	Loop pressure equations	Head loss equations

Contuinity of flow/Convervation of Mass

The total amount of mass flow enter into a node is equal to the mass flow that leave the node plus the flow consumption at the node.

$$\sum \dot{V}_{\text{in}} - \sum \dot{V}_{\text{out}} = \dot{V}_{\text{mass}} \tag{2}$$

Assuming flow consumption is zero,

$$\sum \dot{V}_{\text{in}} - \sum \dot{V}_{\text{out}} = 0 \tag{3}$$

More concretely,

$$A\dot{V} = \vec{0} \tag{4}$$

with the edge flow vector $\dot{V} = \{\dot{V}_1, \dot{V}_2, \dot{V}_z\}$, where z is the number of edges.

Example: Benhassin& Eicker 2013

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix A is the incidence matrix that joins the nodes and the adjacent edges.

Matrix B is a row for every loop and a column for every edge.

Example: Benhassin& Eicker 2013

For node 1, the conservation of mass is stated as

$$\dot{V}_1 - \dot{V}_{11} = 0$$

or

$$A\dot{V} = 0$$

Loop pressure equations

By the **conservation of energy**, the vector of loop pressure residual equals to 0

$$\Delta P = 0 \tag{5}$$

That is, at each loop,

$$\sum_{i=1}^{13} \Delta p_i = 0 \tag{6}$$

In each loop, the pressure losses residual is composed of the pipe pressure losses in the same loop. In other words,

$$\Delta P = B\Delta p \tag{7}$$

Hence we have

$$B\Delta p = \vec{0} \tag{8}$$

Note, the connecting branch currents are independent of each other, as they belong to different meshes. The column of B-matrix indicate which connection branch current flow through the individual branches. That is,

$$\mathrm{B} = B_{ij} = \begin{cases} 1 & \text{if flow in a pipe is the same direction as the definition} \\ -1 & \text{if flow in a pipe is the opposite direction as the definition} \\ 0 & \text{if a pipe is not part of the loop} \end{cases}$$

The total current of a branch results as a linear superposition of these independent currents. More simply,

$$\dot{V} = B^T \dot{V}_L \tag{9}$$

where $\dot{V} = \{\dot{V}_1, \dot{V}_2, \dot{V}_3\}$. The equation dramatically reduces the number of unknown entities required to just three!

Loop pressure equations

Each pipe is considered as a flow resistance whose pressure is proportional to the square of its flow rate. That is,

$$\Delta p = K\dot{V}^2 \tag{10}$$

where K is the vector of resistance coefficient of each pipe given by the Darcy-Weisbech equation.
For every pipe, one gets

$$\Delta p = f(\dot{V}) \tag{11}$$

Hence (8) becomes

$$B \cdot f(\dot{V}) = \vec{0} \tag{12}$$

Substitute (9) into (12) we obtain:

$$B \cdot f(B^T \cdot \dot{V}_L) = \vec{0} \tag{13}$$

Apply the vector function

$$F = f * B^T \tag{14}$$

Equation (14) can be written as

$$F(\dot{V}_L) = \vec{0} \tag{15}$$

The system of nonlinear equation (15) can be solved via using Newton Raphson algorithm in following steps. First, we must define $F(\dot{V}_L)$ and the Jacobien DF.

Nonlinear system and root finding

- A system of nonlinear equation is a set of equations

$$\begin{aligned} f_1(x_1, x_2, \cdot, x_n) &= 0, \\ f_2(x_1, x_2, \cdot, x_n) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \cdot, x_n) &= 0. \end{aligned}$$

- There are three type of nonlinear system in hydraulic model. The solutions are usually found via **Newton-Raphson** or Hardy Cross Method.
- An example of nonlinear system from the DHN in Scharnhauser Park (Hassine and Eicker 2011)

$$\begin{aligned} x^2 - 2x - y + 0.5 &= 0 \\ x^2 + 4 * y^2 - y - 4 &= 0 \end{aligned}$$

- To find the solution (root), we would use the C++ Linear algebra library Eigen. Codes: `https://godbolt.org/z/9NPKAA`

Newtons algorithm

Goal: Starting from initial guess x_1, the Newton Raphson algorithm find the next value of x, i.e. x_{n+1} from previous value x_n.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Scalar case
  1. Input: v_0, eps
  2. Repeat

     $$v_0 = v - f(v)/f'(v)$$

     until $v_n - v \leq$eps
  3. Print v_n, which is the root.

- Vector case
  1. Input: V_0, **eps**
  2. Repeat

     $$V_0 = V - DF(V)^{-1}F(V)$$

     until $V_n - V \leq$eps
  3. Print V_n, which is the root.

Newton iteration:

$$V^{k+1} := V^k - \underbrace{DF(V^k)^{-1}F(V^k)}_{\text{newton correction}},$$

$DF(V^k)$ sufficiently regular

Newton algorithm: C++ implementation (scalar)

```
#include <cmath>
#include <iostream>
using namespace std;

double square_root(double a)
{
    double v = 1.0; // makes an initial guess
    double eps;

    // Iterates using Newton-Raphson recurrence
    while (std::abs(v*v - a)>= eps)
    {
       v = 0.5*(v+a/v);
    }
    //std::cout << v << std::endl;
    return v;
}

int main()
{
    auto v = square_root(1024);
    std::cout << v << std::endl;
}
```

Try it for your self! `https://godbolt.org/z/9adtJE`

We would follow the same method used in electrical case. See circuit.pdf.