"Sapienza" University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Network Algorithms

Lecture notes integrated with the book TODO

Author
Alessio Bandiera

September 29, 2024

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Network Algorithms* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/aflaag-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **alessio.bandiera02@gmail.com**
- LinkedIn: **Alessio Bandiera**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

- Progettazione di Algorithmi

Licence:

1
TODO

1.1 TODO

1.1.1 Classical solutions

Algoritmo 1.1.1.1 *Bellman-Ford*: TODO

```
function BELLMANFORD(G)
    TODO
end function
```

Algoritmo 1.1.1.2 *Dijkstra*: TODO

```
function DIJKSTRA(G)
    TODO
end function
```

Algoritmo 1.1.1.3 *Floyd-Warshall*: Given a directed graph G, and an unconstrained weight function w for the edges, the algorithms returns a matrix `dist` such that `dist`$[u]$$[v]$ is the weight of the least-cost path from u to v.

1: **function** FLOYDWARSHALL(G, w)
2:     Let `dist`$[n][n]$ be an $n \times n$ matrix, initialized with every cell at $+\infty$
3:     **for** $u \in V(G)$ **do**
4:         `dist`$[u][u] = 0$
5:     **end for**
6:     **for** $(u, v) \in E(G)$ **do**
7:         `dist`$[u][v] = w(u, v)$
8:     **end for**
9:     **for** $k \in V(G)$ **do**
10:         **for** $u \in V(G)$ **do**
11:             **for** $v \in V(G)$ **do**
12:                 `dist`$[u][v] = \min($`dist`$[u][k]$, `dist`$[k][v])$
13:             **end for**
14:         **end for**
15:     **end for**
16: **end function**

Idea. The core concept of the algorithm is to construct a matrix using a dynamic programming approach, that evaluates all possible paths between every pair of vertices. Specifically, to determine the shortest path from a vertex u to a vertex v, the algorithm considers two options: either traveling directly from u to v, or passing through an intermediate vertex k, potentially improving the path.

Cost analysis. The `for` loop in line 3 has cost $\Theta(n)$, the `for` loop in line 6 has cost $\Theta(m) = \Theta(n^2)$ and the cost of the triple nested `for` loop is simply $\Theta(n^3)$. Therefore, the cost of the algorithm is

$$\Theta(n) + \Theta(n^2) + \Theta(n^3) = \Theta(n^3)$$

1.2 Interconnection topologies

Up to this point, the routing problem has considered the network as a graph where **the structure is not known to the nodes**, and can change over time due to factors like *faults* and *variable traffic*. However, when the network represents an **interconnection topology**, such as one connecting processors, the structure of the network is known and remains fixed. This characteristic can be leveraged in the packet-routing algorithms.

While the fixed nature of the network topology can be used to develop more efficient routing strategies, efficiency becomes a critical conecrn in interconnection topologies. As a result, solutions with stronger properties than basic shortest-path algorithms are required.

There are many types of routing models. In this notes, the focus will be on the store-and-forward model:

- aata is divided into *discrete packets*;
- each packet contains *control information* (such as source, destination, and sequence data) and is treated as an independent unit that is forwarded from node to node through the network;
- packets may be temporarily stored in **buffer queues** at intermediate nodes if necessary, due to link congestion or busy channels;
- each node makes a **local routing decision** based on the packet's destination address and the chosen routing algorithm;
- during each step of the routing process, **a single packet can cross each edge**;
- additionally, mechanisms for error detection and recovery may be employed to ensure reliable packet delivery, and flow control and congestion management may be applied to optimize network performance.

1.2.1 Bufferfly network

Definition 1.2.1.1: Bufferfly network

Let n be an integer, and let $N := 2^n$; an n-**bufferfly network** is a *layered graph* defined as follows:

- there are $n+1$ layers of N nodes each, for a total of $N(n+1)$ nodes;
- each node is labeled with a pair (w, i), where i is the *layer of the node*, and w is an n-bit binary number that denotes the *row of the node*;
- there are $2Nn = 2 \cdot 2^n \cdot n = n2^{n+1}$ edges;
- two nodes (w, i) and (w', i') are linked by an edge if and only if $i' = i + 1$ and either $w = w'$ (which is a *straight edge*) or w and w' differ in only the i-th bit (which is a *cross edge*).

Example 1.2.1.1 (Bufferfly network)**.** TODO

The nodes of a butterfly are **crossbar switches**, which have two input and two output ports and can operate in two states, namely *cross* and bar (shown below, respectively).

mettere foto

It can be shown that each node (except those in the first and the last layers) has degree 4 . Therefore, $4N$ additional nodes are typically added ($2N$ for the input, and $2N$ for the output) such that $\deg(u) = 4$ for each $u \in V(G)$ — these nodes will not be considered in the networks analyzed in this notes.

prove it?

mettere foto

As a result, a butterfly network can be viewed as a *switching network* that connects $2N$ input units to $2N$ ouptut units, through a layered structure divided into $\log N + 1 = \log 2^n + 1 = n + 1$ layers, each consisting of N nodes.

Moreover, butterfly networks have a recursive structure, which is highlighted in the following figure. Specifically, one n-dimensional butterfly contains two $(n-1)$-dimensional butterfly networks as subgraphs.

aggiungi foto

The topology of the butterfly network can be leveraged as stated in the following proposition.

Proposition 1.2.1.1: Greedy path

Given a pair of rows w and w', there exists a *unique path of length* n, called **greedy path**, from node $(w, 0)$ to node (w', n). This path passes through each layer exactly once, and it can be found through the following procedure:

function GREEDYPATH(w, w')
    for $i \in [1, n]$ **do**
        if $w_i == w'_i$ **then**
            Traverse a *straight edge*
        else
            Traverse a *cross edge*
        end if
    end for
end function