Network Flows

1. Introduction

1.3 Network Representations

Fall 2010

Instructor: Dr. Masoud Yaghini
Network Representations

- The performance of a network algorithm depends:
  - the algorithm
  - the data structures

- **Data structure**
  - The manner used to represent the network within a computer and the storage scheme used for maintaining and updating the intermediate results.
Network Representations

- In representing a network, we need to store two types of information:
  - (1) the network topology, that is, the network's node and arc structure; and
  - (2) data such as costs, capacities, and supplies/demands associated with the network's nodes and arcs.

- Usually the scheme we use to store the network's topology will suggest a natural way for storing the associated node and arc information.
Network Representations

- Node-Arc Incidence Matrix
- Node-Node Adjacency Matrix
- Adjacency Lists
- Forward and Reverse Star Representations
- Compact Forward and Reverse Star Representation
Node-Arc Incidence Matrix
Node-Arc Incidence Matrix

- **Node-arc incidence matrix / incidence matrix**
  - It represents a network as the constraint matrix of the minimum cost flow problem
  - It stores the network as an \( n \times m \) matrix \( N \)
  - It contains one row for each node of the network and one column for each arc.
  - The column corresponding to arc \((i, j)\) has only two nonzero elements: It has a ‘+1’ in the row corresponding to node \(i\) and a ‘-1’ in the row corresponding to node \(j\).
Minimum Cost Flow Problem

- Minimum Cost Flow Problem

Minimize \[ \sum_{(i, j) \in A} c_{ij}x_{ij} \]

subject to

\[ \sum_{\{j: (i, j) \in A\}} x_{ij} - \sum_{\{j: (j, i) \in A\}} x_{ji} = b(i) \quad \text{for all } i \in N, \]

\[ l_{ij} \leq x_{ij} \leq u_{ij} \quad \text{for all } (i, j) \in A, \]

- In matrix form

Minimize \[ cx \]

subject to \[ Nx = b, \]

\[ l \leq x \leq u. \]
Node-Arc Incidence Matrix

\[
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 1 & 1 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 1 & 0
\end{bmatrix}
\]
The node-arc incidence matrix has a very special structure

- Only $2m$ out of its $nm$ entries are nonzero,
- all of its nonzero entries are +1 or -1, and
- each column has exactly one +1 and one -1.
- the number of +1's in a row equals the outdegree of the corresponding node and the number of -1's in the row equals the indegree of the node.
Node-Arc Incidence Matrix

- Because the node-arc incidence matrix $\mathbf{N}$ contains so few nonzero coefficients, the incidence matrix representation of a network is not space efficient.

- More efficient schemes would merely keep track of the nonzero entries in the matrix.

- The node-arc incidence matrix rarely produces efficient algorithms.

- This representation is important because
  - it represents the constraint matrix of the minimum cost flow problem and
  - the node-arc incidence matrix possesses several interesting theoretical properties.
Node-Node Adjacency Matrix
Node-Node Adjacency Matrix

- **Node-node adjacency matrix / adjacency matrix**
  - It stores the network as an $n \times n$ matrix $\mathcal{H} = \{h_{ij}\}$.
  - The matrix has a row and a column corresponding to every node.
  - Its $ij$th entry $h_{ij}$ equals 1 if $(i, j) \in A$ and equals 0 otherwise.
Node-Node Adjacency Matrix

\[
\begin{pmatrix}
1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\]
Node-Node Adjacency Matrix

- If we wish to store **arc costs** and **capacities** as well as the network topology, we can store this information in two additional $n \times n$ matrices $\mathcal{L}$ and $\mathcal{U}$.

- The adjacency matrix has $n^2$ elements, only $m$ of which are nonzero.

- Consequently, this representation is space efficient only if the network is sufficiently dense; for sparse networks this representation wastes considerable space.

- The simplicity of the adjacency representation permits us to use it to implement most network algorithms rather easily.
Node-Node Adjacency Matrix

- We can determine the cost or capacity of any arc 
  \((i, j)\) simply by looking up the \(ij\)th element in the matrix \(L\) or \(U\).
- We can obtain the arcs emanating from node \(i\) by scanning row \(i\)
  - If the \(j\)th element in this row has a nonzero entry, \((i, j)\) is an arc of the network.
- Similarly, we can obtain the arcs entering node \(j\) by scanning column \(j\)
  - If the \(i\)th element of this column has a nonzero entry, \((i, j)\) is an arc of the network.
Node-Node Adjacency Matrix

- These steps permit us to identify all the outgoing or incoming arcs of a node in time proportional to $n$.
- For **dense networks** we can usually afford to spend this time to identify the incoming or outgoing arcs.
- For **sparse networks** these steps might be the bottleneck operations for an algorithm.
Adjacency Lists
Adjacency Lists

We defined before:

- The *arc adjacency list* $A(i)$ of a node $i$ as the set of arcs emanating from that node.
- The *node adjacency list* of a node $i$ as the set of nodes $j$ for which $(i, j) \in A$.

**Adjacency list**

- It stores the *node adjacency list* of each node as a *singly linked list*.
- A *linked list* is a collection of cells each containing one or more fields.
Adjacency Lists

- The node adjacency list for node $i$ will be a linked list having $|A(i)|$ cells and each cell will correspond to an arc $(i, j) \in A$.

- Each cell corresponding to the arc $(i, j)$ will have:
  - One data field will store node $j$.
  - Two other data fields to store the arc cost $c_{ij}$ and the arc capacity $u_{ij}$.
  - One additional link field, which stores a pointer to the next cell in the adjacency list.
    - If a cell happens to be the last cell in the adjacency list, by convention we set its link to value zero.

- We also need an array of pointers that point to the first cell in each linked list.
Adjacency Lists
Forward and Reverse Star Representations
Forward and Reverse Star Representations

- **Forward star representation**
  - It stores the node adjacency list of each node.
  - It is similar to the *adjacency list* representation, but instead of maintaining these lists as linked lists, it stores them in a *single array*.
  - It provides us with an efficient means for determining the set of outgoing arcs of any node.
Forward and Reverse Star Representations

- To develop this representation,
  - We first associate a unique sequence number with each arc, thus defining an ordering of the arc list.
  - We number the arcs in a specific order:
    ◆ first those emanating from node 1, then
    ◆ those emanating from node 2, and so on.
  - We number the arcs emanating from the same node in an arbitrary fashion.
  - We then sequentially store information about each arc in the arc list.
  - We store the tails, heads, costs, and capacities of the arcs in four arrays: \textit{tail, head, cost, and capacity}.
  - If arc \((i, j)\) is arc number 20, we store the tail, head, cost, and capacity data for this arc in the array positions \text{tail}(20), \text{head}(20), \text{cost}(20), \text{and capacity}(20).
Forward and Reverse Star Representations

- A pointer
  - We also maintain a pointer with each node \( i \), denoted by \( \text{point}(i) \), that indicates the smallest-numbered arc in the arc list that emanates from node \( i \).
  - If node \( i \) has no outgoing arcs, we set \( \text{point}(i) \) equal to \( \text{point}(i + 1) \).
  - Therefore, the outgoing arcs of node \( i \) at positions \( \text{point}(i) \) to \( (\text{point}(i + 1) - 1) \) in the arc list.
  - If \( \text{point}(i) > \text{point}(i + 1) - 1 \), node \( i \) has no outgoing arc.
  - For consistency, we set \( \text{point}(1) = 1 \) and \( \text{point}(n + 1) = m + 1 \).
Forward and Reverse Star Representations

```
point | tail | head | cost | capacity
----- |------|------|------|---------
1     | 1    | 2    | 25   | 30      
2     | 1    | 3    | 35   | 50      
3     | 2    | 4    | 15   | 40      
4     | 3    | 2    | 45   | 10      
5     | 4    | 3    | 15   | 30      
6     | 4    | 5    | 45   | 60      
7     | 5    | 3    | 25   | 20      
8     | 5    | 4    | 35   | 50      
```
Forward and Reverse Star Representations

- **Reverse star representation**
  - To determine the set of incoming arcs of any node efficiently

- To develop a reverse star representation
  - We examine the nodes $i = 1$ to $n$ in order and sequentially store the heads, tails, costs, and capacities of the incoming arcs at node $i$.
  - We maintain a reverse pointer with each node $i$, denoted by $rpoint(i)$, which denotes the first position in these arrays that contains information about an incoming arc at node $i$.
  - If node $i$ has no incoming arc, we set $rpoint(i)$ equal to $rpoint(i + 1)$.
  - For consistency, we set $rpoint(1) = 1$ and $rpoint(n + 1) = m + 1$.
  - As before, we store the incoming arcs at node $i$ at positions $rpoint(i)$ to $(rpoint(i + 1) - 1)$. 
Forward and Reverse Star Representations

![Graph]

<table>
<thead>
<tr>
<th></th>
<th>cost</th>
<th>capacity</th>
<th>tail</th>
<th>head</th>
<th>rpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>10</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>30</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>50</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>30</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>50</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>40</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>45</td>
<td>60</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>
Compact Forward and Reverse Star Representation
Forward and Reverse Star Representations

- Observe that by storing both the forward and reverse star representations
  - We will maintain a significant amount of duplicate information.
  - We can avoid this duplication by storing arc numbers in the reverse star instead of the tails, heads, costs, and capacities of the arcs.

- So instead of storing the tails, costs, and capacities of the arcs, we simply store arc numbers;
  - and once we know the arc numbers, we can always retrieve the associated information from the forward star representation.
  - We store arc numbers in an array $trace$ of size $m$. 
Forward and Reverse Star Representations

- Compact forward and reverse star representation

<table>
<thead>
<tr>
<th>point</th>
<th>tail</th>
<th>head</th>
<th>cost</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>15</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>45</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5</td>
<td>45</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>3</td>
<td>25</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>4</td>
<td>35</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>trace</th>
<th>rpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>
Forward and Reverse Star Representations

- As an illustration,
  - arc (3, 2) has arc number 4 in the forward star representation and
  - arc (1, 2) has an arc number 1 in the forward star representation
Forward Star vs. Adjacency List

- **Forward star representation**
  - The major advantage is its space efficiency.
  - It requires less storage than does the **adjacency list** representation.
  - It is easier to implement in languages such as FORTRAN that have no natural provisions for using linked lists.
Node-Node Adjacency Matrix

- **Adjacency list**
  - The major advantage is its ease of implementation in languages such as C or Java that are able to manipulate linked lists efficiently.
  - Further, using an adjacency list representation, we can add or delete arcs (as well as nodes) in constant time.
  - On the other hand, in the forward star representation, these steps require time proportional to $m$, which can be too time consuming.
## Summary

<table>
<thead>
<tr>
<th>Network representations</th>
<th>Storage space</th>
<th>Features</th>
</tr>
</thead>
</table>
| Node–arc incidence matrix     | $nm$                   | 1. Space inefficient  
2. Too expensive to manipulate  
3. Important because it represents the constraint matrix of the minimum cost flow problem |
| Node–node adjacency matrix    | $kn^2$ for some constant $k$ | 1. Suited for dense networks  
2. Easy to implement          |
| Adjacency list                | $k_1n + k_2m$ for some constants $k_1$ and $k_2$ | 1. Space efficient  
2. Efficient to manipulate  
3. Suited for dense as well as sparse networks |
| Forward and reverse star      | $k_3n + k_4m$ for some constants $k_3$ and $k_4$ | 1. Space efficient  
2. Efficient to manipulate  
3. Suited for dense as well as sparse networks |
The End