IN7K3: Flujo en Redes

3: Maximum Flow

Lecturer: Fernando Ordóñez

1 Introduction

For a given directed network, we want to determine the maximum amount of flow that can be transported from a source node s to a sink node t.

$$\begin{aligned} \max \quad & v \\ \text{s.t.} \quad & \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = \begin{cases} v & \text{if } i = s \\ 0 & \text{if } i \neq N \setminus \{s,t\} \\ -v & \text{if } i = t \end{cases} \\ & 0 \leq x_{ij} \leq u_{ij} \end{aligned}$$

We assume: u_{ij} are non-negative integers, there is no ∞ capacity path from s to t, no parallel arcs.

1.1 Feasible Flow Problem

Suppose there is merchandise available at certain ports and a demand for this merchandise at other ports. Given a network between these supply and demand nodes with capacities on the arcs, can we satisfy this demand?

1.2 Scheduling on Uniform Parallel Machines

Consider the problem of scheduling a set of J production jobs on M uniform parallel machines. Each job $j \in J$ has a processing requirement p_j (denoting the time a machine requires to complete the job), a release date r_j (when job j is available for processing), and a due date $d_j \geq r_j + p_j$. We assume that a machine can work on one job at a time, that a job can be at one machine at a time, and that jobs can be preempted (stop a job and continue later from where it was left.) Find a schedule of the jobs on the machines that completes all jobs in time, or show that one does not exists. Consider the example with $M = 3$ and the following jobs and requirements:

Job (j)	1	2	3	4
Processing time (p_j)	1.5	1.25	2.1	3.6
Release time (r_j)	3	1	3	5
Due date (d_j)	5	4	7	9

2 Flows and Cuts

For any two nodes $i, j \in N$ we define an $i - j$ cut as:

The capacity of $i - j$ cut is:

Proposition 1. *The value of any flow is less than or equal to the capacity of any $s - t$ cut in the network*

Residual Network

For a given flow x on the network we define the residual network $G(x)$ by:

- Same set of nodes
- For each arc (i, j), we get two arcs on $G(x)$: an (i, j) with capacity $u_{ij} - x_{ij}$ and cost c_{ij}, and an arc (j, i) with capacity x_{ij} and cost $-c_{ij}$.
- For simplicity we denote the capacity arcs in $G(x)$ by $r_{ij} = u_{ij} - x_{ij} + x_{ji}$. ($x_{ji}$ accounts for sending back the flow on arc (j, i)).

For example

3 Max-Flow Min-Cut Theorem

Theorem 2. *(Optimality conditions for max flow/ Max-flow Min-cut) The following statements are equivalent:*

1. *A flow x is maximum*
2. *There is no augmenting path in $G(x)$*
3. *There is an $s-t$ cut $[S, \bar{S}]$ with capacity $u([S, \bar{S}])$ equal to the flow value of x.*

Proof: $(1 \Rightarrow 2)$

$(3 \Rightarrow 1)$

$(2 \Rightarrow 3)$

3.1 Consequences of Max Flow - Min Cut

Network Connectivity. A communications network might be interested in the maximum number of arc disjoint paths from s to t. (i.e. if an arc fails s and t remain connected)

Theorem 3. *For $G = (N, A)$ a directed graph. The maximum number of arc-disjoint paths from s to t equals the minimum number of arcs whose deletion disconnects s from t*

4 Generic Augmenting Path Algorithm

An *augmenting path* is a directed path from s to t in the residual network $G(x)$.

```
Set $x = 0$
while $G(x)$ contains a directed path from $s$ to $t$ do
    Find an augmenting path $P$ in $G(x)$
    $\delta = \min\{r_{ij} \mid (i, j) \in P\}$
    augment the flow along $P$ by $\delta$ and update $G(x)$
endwhile
```

4.1 Correctness of Augmenting Path Algorithm

This algorithm is finite because:

It finishes with the optimal solution because:

4.2 Complexity of Augmenting Path Algorithm

Find augmenting path in $G(x)$, we run a search (breadth/depth) algorithm, this takes:

At each iteration we increase the amount of flow by:

An upper bound on the total flow is:

Total Complexity:

5 Labeling Algorithm

```
Set $x = 0$, label node $t$
while node $t$ is labeled do
    Unlabel all nodes
    Use search algorithm to label all nodes reachable from $s$ in $G(x)$
    if node $t$ is labeled
        Find an augmenting path $P$ in $G(x)$
        $\delta = \min\{r_{ij} \mid (i,j) \in P\}$
        augment the flow along $P$ by $\delta$ and update $G(x)$
    endif
endwhile
```

A very bad example....

Total complexity of Labeling Algorithm $O(mnU)$.

Improvements on this algorithm can be made by simple rules on how to select the path P from s to t in $G(x)$ at each iteration.

The largest augmenting path algorithm: Select P with maximum δ.

The capacity scaling algorithm: Select path P based on a certain threshold Δ

```
$\Delta$-scaling phase: For a fixed value of $\Delta$
    Select $P$ with capacity at least $\Delta$
    If no such $P$ exists, set $\Delta = \Delta/2$
```

The shortest augmenting path algorithm: Select P with the fewest number of arcs.

6 Capacity Scaling Algorithm

For any Δ, we define $G(x, \Delta)$ as:
- same set of nodes of $G(x)$ (and N)
- the arcs in $G(x)$ of capacity at least Δ

```
Set $x = 0$, $\Delta =$        (large)
while $\Delta \geq 1$ do
    $\Delta = \Delta/2$
    while there is a path from $s$ to $t$ in $G(x, \Delta)$ do
        Find an augmenting path $P$ in $G(x, \Delta)$
        $\delta = \min\{r_{ij} \mid (i, j) \in P\}$
        augment the flow along $P$ by $\delta$ and update $G(x, \Delta)$
    endwhile
endwhile
```

Complexity of Capacity Scaling

There are $O(\log U)$ scaling phases:

The running time per scaling phase is $O(m^2)$:

The total running time is:

7 Shortest Path Algorithm

This modification of Labeling/Ford-Fulkerson algorithm maintains distance labels on the nodes to look for the shortest path P in $G(x)$ at each iteration.

A set of distance labels $d(i)$, for $i \in N$ is said to be valid if
$d(t) = 0$
$d(i) \leq d(j) + 1$ for every $(i, j) \in G(x)$

If $d(i) = d(j) + 1$ for an arc $(i, j) \in G(x)$, then we say arc (i, j) is admissible.

Example:

Lemma 4. *If there is an admissible path P from s to t then it is a shortest path*
proof:

```
Set $x = 0$
Set $d(i) =$ shortest path from $i$ to $t$ (in # of arcs)
while $d(s) < n$ do
    if there is $j \in N \setminus \{t\}$ with no outgoing admissible arc
        $d(j) = 1 + \min\{d(k) \mid (j,k) \in G(x)\}$
    else
        find an admissible path $P$ from $s$ to $t$ and augment $x$
    endif
endwhile
```

Complexity of shortest augmenting path

If $d(i)$ are valid distance labels, then they lower bound the no. of arcs from i to t in $G(x)$

When we update $d(i)$, the distance labels remain valid, and $d(i)$ strictly increases

If $d(i) \geq n$ there is no path from i to t

Total number of updates of $d(i)$. Total number for all $i \in N =$.

At each iteration one admissible arc is saturated

A saturated (i,j) is inadmissible until flow is sent on (j,i), which implies $d(j)$ increased

An arc can be saturated times. Total number of augmentations is .

We can determine an augmentation in $O(n)$ time. Total complexity =

8 Matchings and Covers

A *matching* is a subset of A such that no two arc are incident to a common node. The **Matching Problem:** What is the maximum cardinality matching?

A *cover* is a subset of N such that every arc in A is incident to some node in the cover. The **Min Cover Problem:** What is the minimum cardinality cover? Consider a bipartite

network $G = (N, A)$. That is N can be partitioned into $N1$ and $N2$ such that $A \subset (N1 \times N2) \cup (N2 \times N1)$.

Theorem 5. *In a bipartite network, the maximum matching is equal to the minimum cover*

proof:

Given a bipartite network $G = (N_1 \cup N_2, A)$, we define the following max flow network by, setting capacities on the arcs in A to be ∞, adding a source node s and arcs of capacity one from s to every node in N_1, and arcs of capacity one from every node in N_2 to a sink node t.

Let M be a matching and K be a cover of G.

- There is a flow of value $= |M|$.

  For every $(i, j) \in M$ send one unit of flow on the path $s - i - j - t$. Since M is a matching this will route at most a single unit of flow through every node. This flow has a value $= |M|$.

- There is a cut of capacity $= |K|$.

  We can separate node s and t in the graph by removing the following K arcs, each of capacity 1: for each node $k \in K$, if $k \in N_1$ remove (s,k) if $k \in N_2$, remove (k,t). The set of nodes reachable from s forms a cut S, of capacity $u([S,\bar{S}]) = |K|$.

  By taking the maximum cardinality matching and the minimum cardinality cover we obtain that
  $$\max|M| \leq \max \text{ flow } = \min \text{ cut } \leq \min|K| \ .$$

- For every cut $(S,\bar{S})$ of finite capacity there is a cover such that $u([S,\bar{S}]) = |K|$.

  The cut only includes arcs of the form (s,i) or (j,t) because it has finite capacity. Construct a set K with every node of the original graph incident to an arc in the cut. The set K forms a cover. If not, there exists an arc $(k,l) \in A$ with $k \notin K$ and $l \notin K$. Then the graph still includes a path $s-k-l-t$ and S is not a cut.

- The maximum flow of value v induces a matching s.t. $|M| = v$.

  Run the augmenting path algorithm to obtain the maximum flow of value v. Since all the capacities are integer (we can consider the infinite capacities some very large integer numbers without loss of generality), every augmentation will send an integer amount of flow, in fact will send exactly 1 unit of flow. A consequence is that the maximum flow obtained will send unit flows on the original arcs. The arcs used then constitute a matching of cardinality v.

9 Networks with Lower Bounds

Example: If l_{ij} can be positive, then the max flow problem can be infeasible.

The new residual network:

The new residual capacity:

The capacity of cut $[S, \bar{S}]$:

Theorem 6. *(Generalized Max-Flow Min-Cut Theorem) In a network with both lower and upper bounds on arc flows, the maximum value of flow from node s to node t equals the minimum capacity among all $s - t$ cuts.*

Theorem 7. *(Flow Feasibility Conditions) A network with non-negative lower bounds has a feasible flow if and only if for every set S of nodes*

$$\sum_{(i,j)\in(\bar{S},S)} l_{ij} \leq \sum_{(i,j)\in(S,\bar{S})} u_{ij} \ .$$