Network Design and Optimization course

Lecture 2

Alberto Ceselli
alberto.ceselli@unimi.it

Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano

October 4th, 2011

The problem

Given

- an existing network

I want to build a **routing table**, that is

- decide which links to use (route) for connecting each pair of nodes
- maximizing the overall quality of service (e.g. minimizing delay or power loss)

Assumptions

Some assumptions:

1. all packets must be routed on the same links,
2. the capacity of each link is enough for any connection request,
3. connections using the same links at the same time do not interfere.

N.B. imposing (1), or assuming that link usage cost does not depend on the amount of traffic routed is the same; conditions (2) and (3) are linked when using packet routing.

Recognizing a known problem ...

We are facing an *All Pairs Shortest path problem*!

Modeling the costs

Step 1: estimating link usage costs. How?

- Know your network;
- make suitable assumptions and simplifications!

(see examples from previous slides).

Network model

Given a network, build a graph $G = (V, E)$ having

- one vertex $i \in V$ for each node of the network
- one edge $e \in E$ for each link of the network
- costs c_e on each arc $e \in E$
- prizes g_v on each node $v \in V$
- a special vertex $s \in V$ representing the origin of packets
- a special vertex $t \in V$ representing the destination of packets

Question for you: how to find shortest paths between all pairs of vertices in a graph?

Floyd Warshall algorithm

```
int cost[][];

// initialize cost[i][j] to c(i,j)

procedure FloydWarshall ()
   for k := 1 to |V|
      for i := 1 to |V|
         for j := 1 to |V|
            cost[i][j] =
               min(cost[i][j],cost[i][k]+cost[k][j]);
```

Floyd Warshall correctness proof

Theorem: *FW returns the shortest path matrix*
Proof: By invariants

.

Floyd Warshall correctness proof

Theorem: *FW returns the shortest path matrix*
Proof: By invariants
Before iteration k, cost[i][j] is the cost of the *shortest path* connecting i and j, and using only vertices in $1 \ldots k$ (besides i and j themselves).

Floyd Warshall correctness proof

By induction

- base: at step 1 cost[i][j] = c(i,j)
- step:
  -
  -
  -
  -

Floyd Warshall correctness proof

By induction

- base: at step 1 cost[i][j] = c(i,j) $\rightarrow$ OK!
- step:
  -
  -
  -
  -

Floyd Warshall correctness proof

By induction

- base: at step 1 cost[i][j] = c(i,j) $\rightarrow$ OK!
- step:
  - at step k, cost[i][j] is the SP among i and j using vertices $1 \dots k$
  -
  -
  -

Floyd Warshall correctness proof

By induction

- base: at step 1 cost[i][j] = c(i,j) $\rightarrow$ OK!
- step:
  - at step k, cost[i][j] is the SP among i and j using vertices $1 \dots k$
  - case (1): at step $k+1$, cost[i][j] is also the SP using vertices $1 \dots k+1$ (no update)
  -
  -

Floyd Warshall correctness proof

By induction

- base: at step 1 cost[i][j] = c(i,j) $\rightarrow$ OK!
- step:
  - at step k, cost[i][j] is the SP among i and j using vertices $1 \dots k$
  - case (1): at step $k+1$, cost[i][j] is also the SP using vertices $1 \dots k+1$ (no update)
  - case (2): at step $k+1$, cost[i][j] is obtained going from i to $k+1$ (using only vertices $1..k$) and from $k+1$ to j (same thing)
  -

Floyd Warshall correctness proof

By induction

- base: at step 1 cost[i][j] = c(i,j) $\rightarrow$ OK!
- step:
  - at step k, cost[i][j] is the SP among i and j using vertices $1 \dots k$
  - case (1): at step $k+1$, cost[i][j] is also the SP using vertices $1 \dots k+1$ (no update)
  - case (2): at step $k+1$, cost[i][j] is obtained going from i to $k+1$ (using only vertices $1..k$) and from $k+1$ to j (same thing)
  - $\rightarrow$ only vertices $1 \dots k+1$ are involved.

Floyd Warshall rebuild path

```
// init cost[i][j] to c(i,j) and next[i][j] to 'null'
procedure FloydWarshallWithPathReconstruction ()
   for k := 1 to |V|
      for i := 1 to |V|
         for j := 1 to |V|
            if cost[i][k] + cost[k][j] < cost[i][j] then
               cost[i][j] := cost[i][k] + cost[k][j];
               next[i][j] := k;

procedure Path(i,j)
   if cost[i][j] equals infinity then return "NoPath";
   if next[i][j] equals 'null' then
     return " "; /* no vertices between i and j */
   else
     return
       Path(i,next[i][j])+next[i][j]+Path(next[i][j],j);
```

Floyd Warshall algorithm: complexity

```
int cost[][];

// initialize cost[i][j] to c(i,j)

procedure FloydWarshall ()
   for k := 1 to |V|
      for i := 1 to |V|
         for j := 1 to |V|
            cost[i][j] =
               min(cost[i][j],cost[i][k]+cost[k][j]);
```

Floyd Warshall algorithm: complexity

```
int cost[][];

// initialize cost[i][j] to c(i,j)

procedure FloydWarshall ()
   for k := 1 to |V|
      for i := 1 to |V|
         for j := 1 to |V|
            cost[i][j] =
               min(cost[i][j],cost[i][k]+cost[k][j]);
```

Complexity $O(|V|^3)$

Johnson's Algorithm

See Orlin's slides

All pairs shortest paths algorithm implementation

Let's compute all pair shortest paths algorithms in AMPL ...