Routing

• Topics:
  ○ Definition
  ○ Architecture for routing
    – data plane algorithm
  ○ Current routing algorithm
    – control plane algorithm
  ○ Optimal routing algorithm
    – known algorithms and implementation issues
    – new solution
  ○ Routing mis-behavior: selfish routing
    – price of anarchy or how bad can it be?
Routing

• Definition
  ○ The task of determining how data should travel from its source to destination in the network so that
    – overall delay is minimized (or user utility is maximized), and
    – network resource utilization is maximized

• An example
  ○ Use of “MapQuest.com” to find “driving directions”
    – data plane algorithm

• Ideally, MapQuest.com should be designed so that
  ○ Traffic directions are created such that overall congestion on roads is minimized
  ○ Overall social satisfaction by using such directions is maximized
    – control plane algorithm
Routing

• Essential requirements for routing in a network
  o Unique addressing mechanism in the network,
    – such as, street address
  o Network topology,
    – for example, road map
  o Or, at least local directional information,
    – e.g. information of the form: left on Vassar St. from Main St. will connect it to Mass Ave

• Next, we’ll consider Internet
Routing: Internet

- Internet
  - All nodes have unique IP address
    - allocated according to certain criteria, such as
    - all MIT IP address are of the form 128.31.*.*
  - The topological information is stored in the form of "next-hop" information
    - routing-tables stored in local gateways
  - When data needs to be routed from source to destination,
    - "network layer" takes care of routing
    - converts web address info IP, e.g
    - www.yahoo.com → 142.226.51
    - appends destination IP to packet
    - determines next-hop using routing information of gateway
    - intermediate gateways forward packet using it’s destination IP and their routing table
Example: Routing in Internet

An example of data routing in Internet using routing table information

Q: How to configure these routing tables?
Routing: Control Algorithm

- The control part of routing is about configuring routing tables
- Given a network, traffic demands the routing determines
  - Network utilization or throughput
  - Average delay or utility
- Example

Route 1:
A sends all data to D via X
B sends all data to D via L
Thput = \( \frac{5+20}{10+20} = \frac{25}{30} \)

Route 2:
A sends half-data to D via X and other half via L
B sends all data to D via L
Thput = 1
Routing: Control Algorithm

- Formally, the problem of routing is as follows:
  - Each gateway needs to decide what fraction of data destined for certain destination needs to go through which of its outgoing link
  - So as to maximize overall network throughput and minimize end-to-end delay
  - The parameters of this problem are:
    - network topology and link capacities
    - traffic demand
  - Constraints: decision of routing must be done in distributed manner and should be robust against few failures

- Next, we’ll see how to model the problem of routing
  → It will lead to appropriate algorithm design
Routing: Problem Formulation

• Let nodes of network be numbered $1, \ldots, n$.

• Let $\mathcal{L} = \{(i, j) : 1 \leq i, j \leq n\}$ be set of all links

  ○ $C_{ij}$ be capacity of link $(i, j)$

  ○ $C_{ij} = 0$, if link $(i, j)$ is not present

• Let $r_i(j)$ be rate at which data is generated at node $i$ for node $j$.

• Let $\phi_{\ell i}(j)$ be fraction of data arriving at node $\ell$, destined for node $j$, that is routed to node $i$.

• Let $t_i(j)$ be net data arriving at node $i$ destined for $j$. These satisfy the following relation

$$t_i(j) = r_i(j) + \sum_{(\ell, i) \in \mathcal{L}} t_{\ell}(j) \phi_{\ell i}(j) \quad \text{for all } i, j$$
Routing: Problem Formulation

- Let $F_{i\ell}$ rate at link $(i, \ell) \in \mathcal{L}$
  \[ F_{i\ell} = \sum_{k} t_{k}(k) \phi_{i\ell}(k) \]

- $D_{i\ell}(F_{i\ell})$: delay as func. of $F_{i\ell}$
  - Let it be convex, increasing, twice-differentiable
  - $D_{i\ell}(0) = 0$; $D_{i\ell}(C_{i\ell}^+) = \infty$

- In general, the delay can be replaced by appropriate utility function

- Given this setup, next we describe the question of optimal routing
Optimal Routing

Route-opt:

\[
\min \sum_{(i, \ell) \in \mathcal{L}} D_{i\ell}(F_{i\ell})
\]

subject to

\[
F_{i\ell} = \sum_k t_k(k) \phi_{i\ell}(k); \]

\[
t_k(j) = r_k(j) + \sum_\ell t_\ell(j) \phi_{\ell i}(j); \]

\[
\phi_{\ell i}(j) \geq 0, \text{ for all } i, j, \ell; \]

\[
\sum_i \phi_{\ell i}(j) = 1, \text{ for all } \ell, j.
\]

• An implicit constraint is \(F_{ij} < C_{ij}\), as otherwise optimal cost will be \(\infty\).
  
  o If \(r_i(j)\) are s.t. they can not be routed, then optimal cost will be \(\infty\)

• Before considering algorithms for Route-opt, let’s look at an example.
Example: Route-opt

- Capacity of all links = 5
- Delay function for all links, \( D(x) = \frac{x}{5-x} \)

A feasible routing:

- \( \phi_{12}(4) = 0.5 \); \( \phi_{13}(4) = 0.5 \); \( \phi_{34}(4) = 1 \); \( \phi_{24}(4) = 1 \)
- \( t_1(4) = 2 \); \( t_2(4) = 3 + 0.5 \times 2 = 4 \); \( t_3(4) = 1 + 0.5 \times 2 = 2 \)
- \( F_{12} = 1 \); \( F_{13} = 1 \); \( F_{24} = 4 \); \( F_{34} = 2 \)
- Total delay

\[
\frac{1}{5-1} + \frac{1}{5-1} + \frac{4}{5-4} + \frac{2}{5-2} \approx 5.17
\]

- Next, some algorithms
• Consider following heuristic for Route-opt
  o Assign weights to edges, were weight reflects delay
  o An example, weight = \( \frac{1}{\text{capacity}} \).
• Then, route with minimal delay between a pair of nodes corresponds to minimum weighted path (shortest path).
  o Routing algorithm is equivalent to finding shortest path between node-pairs
• Currently in the Internet
  o (A version of) Shortest path routing (OSPF) is used
  o Weights are based on certain heuristic utilizing observed link quality
• Next, we describe algorithms for finding shortest path
Dijkstra’s Algorithm

• Main idea:
  ◦ iteratively find shortest path between nodes
    – with paths of increasing length, starting with 1

• Algorithm: Find shortest path from node 1 to all nodes

1. Initially, $P = \{1\}$, $D_1 = 0$, $D_j = d_{ij}$, $j \neq 1$
   $[d_{ij}$: weight of edge $(i, j)$; $d_{ij} = \infty$ if not connected $]$
2. Find next closest node: find $i \notin P$ s.t.
   $$D_i = \min_{j \notin P} D_j ;$$
   Set $P = P \cup \{i\}$. If $P$ contains all nodes, then STOP.
3. Update: For all $j \notin P$, set
   $$D_j = \min[D_j, d_{ji} + D_i]$$
4. Go to (2)
Dijkstra’s Algorithm

- The Dijkstra’s algorithm is totally distributed
  - It can also be implemented in parallel and
  - Does not require synchronization

- In the algorithm
  - $D_j$ can be thought of as estimate of shortest path length between 1 and $j$ during the course of algorithm

- The algorithm is one of the earliest example of graph algorithms
  - Reference: Chapter 5.2, Bertsekas and Gallager

- Next, we present the proof of correctness of algorithm
Dijkstra’s Algorithm

• We first state the following two properties of the Algorithm
  
  ◦ **Claim 1.** \( D_i \leq D_j \quad \forall i \in P \, ; \quad j \notin P \)
  
  ◦ **Claim 2.** \( D_j \) is, for each \( j \), the shortest distance between \( j \) and 1, using paths whose nodes all belong to \( P \) (except, possibly, \( j \))

• Given the above two properties
  
  ◦ When algorithm stops, the shortest path lengths must be equal to \( D_j \), for all \( j \)
    
    \( \rightarrow \) That is, algorithm finds the shortest path as desired

• Next, we prove these two claims.

• **Proof of Claim 1**
  
  ◦ The proof follows by simple Induction
    
    – initially Claim 1 is true, and
    
    – always remains true under the update rule
Dijkstra’s Algorithm

• Proof of Claim 2

  ○ We will prove by Induction
  ○ Initially, \( P = \{1\} \) and holds for \( D_1 = 0 \).
  ○ Induction hypothesis
    – let it be true for all nodes till some iteration
  ○ Next, node \( i \) is added to \( P \)
    – let \( D_k \) be distances of nodes before \( i \) was added
    – let \( D'_k \) be distances of nodes after \( i \) was added
  ○ Note that Claim 2 holds for all \( j \in P \) from induction hypothesis as their \( D_j = D'_j \).
  ○ For \( j = i \), \( D_i = D'_i \) satisfies the desired claim from induction hypothesis as well.
Dijkstra’s Algorithm

• Let \( j \notin P \cup \{i\} \). Let \( \hat{D}_j \) be shortest distance from \( j \) to 1 along path containing nodes in \( P \cup \{i\} \).
  
  ○ Let this path have arc \((j, k)\); \( k \in P \cup \{i\} \)
  
  ○ Then, \( \hat{D}_j = \min_{k \in P \cup \{i\}} \left[d_{jk} + D_k\right] \) [induction hypothesis]
    
    \[ = \min \left[ \min_{k \in P} [d_{jk} + D_k], d_{ji} + D_i \right] \]
  
  ○ By induction hypothesis, \( D_j = \min_{k \in P} [d_{jk} + D_k] \)
  
  ○ Hence, \( D_j = \min [D_j, d_{ji} + D_i] \)
    \[ = D'_j \text{(by update of algorithm)} \]

• This completes the proof of Claim 2.
Other Algorithms

• Another popular algorithm
  ○ Bellman–Ford algorithm based on standard “Dynamic Programming”

• The Dijkstra’s algorithm has very efficient distributed implementation
  → Hence, popular

• Current Internet protocols use some version of Dijkstra’s algorithm

• Unfortunately, this algorithm does not perform very well
  ○ Easy to construct examples
  ○ Poor performance is often experienced in practice
Optimal Routing

• Why heuristic based on shortest path?
  ○ Poor in performance, but
  ○ Easy to implement, distributed
  ○ Robust against failures in network
  ○ Quickly adaptive
  ○ More importantly, allows heterogeneous networks (ISP) to operate without sharing “sensitive” information

• We’ll look at the Route-opt problem
  ○ First, we’ll see a non-implementable solution
  ○ Then, look for an implementable solution
     – very similar to Dijkstra’s algorithm
Route-opt

- Convex optimization (minimization) problem
  - Convex cost function
  - Convex constraint set
  → Known-standard methods to solve the problem iteratively

- Let’s look at a simple method called *Descent* method
  - Essentially, it changes the solution iterative so that
  - Solution remains feasible and the cost decreases

- Later, we’ll see a simple, distributed algorithm
  - An extension of descent method
    - Subgradient method via dual decomposition
Descent Method

• Main steps:

1. Start with any initial feasible solution
2. Given current solution, find increment in solution
   – that will retain feasibility of solution and decrease cost.
3. Repeat 2 until convergence.

• Questions:

A. How to find feasible solution initially?
B. What guarantees existence of a feasible “descent” direction?
C. How to find feasible descent direction?
D. What guarantees that convergent point is optional solution?

• Next, we answer these questions
A. Feasible Solution

• It is sufficient to know the method to route data destined for one node
  ○ Repeating it for all destinations gives a complete solution

• Parameters:
  ○ $R_\tau$: residual graph at end of iteration $\tau$
    - Initially, $R_0 = G (n \text{ nodes, } \mathcal{L} \text{ links, } C_{ij} \text{ capacities})$
    - $R_\tau$ changes over time only in capacities
  ○ Traffic demands $r_1(n), \ldots, r_{n-1}(n)$

• Find-Path$\left( R_1, (i, n) \right)$
  ○ Finds a path from $i \rightarrow n$ in $R$ with some positive capacity
  ○ It identifies path and capacity that it has between $i \rightarrow n$
  ○ An important building block of method for finding feasible flow
A. Feasible Solution

(i) Initially: $\tau = 0$, $R_0 = G$, $i = 1$ and $r = r_1(n)$

(iii) Find-path\(\left( R_\tau, (i, n) \right)\) returns path \((i, a_1, \ldots, a_{\ell_1}, n)\) with capacity $c_\tau$

- $q = (r - c_\tau)^+$
- Update $R_\tau$ to obtain $R_{\tau+1}$ as follows:
  - Reduce capacities on edges $(i, a_1), (a_1, a_2), \ldots, (a_{\ell_1}n)$ by $q$
  - Increase link capacities on edges $(a_1, i), (a_2, a_1), (a_3, a_2), \ldots, (n_1a_\ell)$ by $a$

(c) $\tau = \tau + 1$, $i = i + 1$

(iv) If $i > n$ then STOP; else set $r = r_i(n)$

(v) If $r > 0$, go to (iii); else set $i = i + 1$, go to (iv)
A’. Find-Path

- Essentially, probe all directions in given graph in an intelligent way!

- Find-Path\((R, (i, n))\):

  (finds path in graph \(R\) from \(i\) to \(n\) with positive capacity)

  1. Initially \(N = \{i\}\) ; \(c(i) = \infty\) ; \(P = \text{empty}\)
  2. (Add new node)
     - Find \(j \not\in N\) s.t. \(\exists k \in N\) with \(c_{kj} > 0\)
     - Add \(P = P \cup \{(k, j)\}\) ; \(N = N \cup \{j\}\)
     - Set \(c(j) = \min\{c(k), c_{kj}\}\)
  3. If \(n \in N\) then STOP
  4. Path found is of capacity \(c(n)\) with edge in \(P\) (there is such a unique path)
B. Existence of Descent Direction

• Recall that a feasible routing is characterized by $\phi = (\phi_{k\ell})$
  
  ◦ By definition, set of all feasible $\phi$ is convex

• Claim. $\phi$ is optimal $\iff$ No feasible descent direction.

• Proof.
  
  ◦ First, direction ($\Rightarrow$) which we prove by contradiction
    
    – let $\phi$ be optimal and $\exists$ a feasible decent direction
    
    – that is, there exists $\Delta \phi$ such that
      
      – $\phi + \theta \Delta \phi$ is feasible and
      
      – $D(\phi + \theta \Delta \phi) < D(\phi)$ for some $\theta > 0$
      
      – this contradicts assumption of $\phi$ being optimal

  ◦ Thus $\phi$ is optimal then no feasible descent direction
B. Existence of Descent Direction

• Next, we prove the other direction ($\Leftarrow$):
  ○ Equivalently, if $\phi$ is not optimal then
    – there exists a feasible descent direction

• Let $\phi$ not be optimal
  ○ That is, there is $\hat{\phi}$ feasible s.t. $D(\hat{\phi}) < D(\phi)$

• Let
  \[ \phi_\theta = \phi + \theta(\hat{\phi} - \phi) = (1 - \theta)\phi + \theta\hat{\phi} \; ; \; \theta \in (0, 1) \]

• Then, $\phi_\theta$ is feasible due to convexity of feasible set.
  \[ D(\phi_\theta) \leq \theta D(\hat{\phi}) + (1 - \theta)D(\phi) < D(\phi) \; ; \; \theta \in (0, 1) \]

• Thus, $(\hat{\phi} - \phi)$ is a feasible descent direction.

• Thus, $\phi$ not opt. $\Rightarrow$ existence of feasible descent direction.

• This complete the proof of our claim $\square$
C. Finding Descent Direction

• First, suppose we are in unconstrained set up
  • Let $\nabla D(\phi) = \begin{bmatrix} \frac{\partial D(\phi)}{\partial \phi_k} \end{bmatrix}^T_{k,\ell}$
  • Then

• Claim. $-\nabla D(\phi)$ is a descent direction.

• Proof.
  • $D$ is assumed to be a strictly convex, twice differentiable function
    — that is, Hessian of $D$ is strictly positive
  • Let, $\phi(t) = \phi - t\nabla D(\phi)$
  • By Taylor’s expansion,

$$D\left(\phi(t)\right) = D(\phi) + t(\phi(t) - \phi)^T \nabla D(\phi) + \frac{t^2}{2} (\phi(t) - \phi)^T \nabla^2 D(\phi(s)) (\phi(t) - \phi)$$

for some $s \in (0, t)$. 
C. Finding Descent Direction

- Let $\nabla^2 D(\phi(s)) \leq MI$; $s \in (0, t)$
- Then,
  \[
  D(\phi(t)) \leq D(\phi) - t\|\nabla D(\phi)\|_2^2 + \frac{t^2}{2}M\|\nabla D(\phi)\|_2^2
  \]
- Then, for $t < 2/M$; $D(\phi(t)) < D(\phi)$, if
  \[-\|\nabla D(\phi)\|_2^2 \neq 0.\]
Gradient Descent

- **Algorithm**
  - Start from some feasible $\phi$.
  - Set $\nabla \phi = -\nabla D(\phi)$.
  - Find $t \in [0, 1]$ s.t. $D(\phi + t\nabla \phi)$ is minimum.
  - Set $\phi = \phi + t\nabla \phi$.
  - Repeat from #2 until $\nabla D(\phi) = 0$.

- **Main Problem**
  - Above works for unconstrained setup.
  - What about constrained situation?
    - “project” gradient descent into feasible space
    - we’ll consider a modification of this specialized to Route-Opt setup
C. Finding Descent Direction

- Given $\phi$, there exists $\Delta \phi$ such that
  - $\phi + \Delta \phi$ is feasible and
  - $D(\phi + \Delta \phi) < D(\phi)$

- Given this, for all $\theta \in (0, 1)$,
  \[
  \phi(\theta) = \theta(\Delta \phi + \phi) + (1 - \theta)\phi,
  \]
  is feasible by convexity,
  \[
  D\left(\phi(\theta)\right) < D(\phi) ; \forall \theta \text{ by convexity.}
  \]

- For very small $\theta$, $D\left(\phi(\theta)\right) \approx D(\phi) + \theta \cdot \nabla D(\phi)^T \Delta \phi$,
  - Then, $\nabla D(\phi)^T \Delta \phi < 0$

- Thus, one option is to look for $\Delta \phi$ such that
  - $\phi + \Delta \phi$ remains feasible and
  - $\nabla D(\phi)^T \Delta \phi < 0$

- We do this next.
C. Finding Feasible Descent Direction

- Let $\Delta \phi$ be such that $\phi + \Delta \phi$ is feasible, i.e.
  - Re-routing certain data for destination $n$ from some $i$ along other path

- Hence, descent direction means that for some $(i, n)$ pair $\exists$ two paths
  
  $P_1 = (i, a_1, \ldots, a_\ell, n)$
  
  $P_2 = (i, b_1, \ldots, b_k, n)$

- Such that
  - $P_1$ has positive flow on $P_1$ from $i \to n$;
  - $P_2$ has some capacity left for $i \to n$ and
    
    $\frac{\partial D(\phi)}{\partial \phi_{ia_1}} + \cdots + \frac{\partial D(\phi)}{\partial \phi_{a_\ell n}} > \frac{\partial D(\phi)}{\partial \phi_{b_1}} + \cdots + \frac{\partial D(\phi)}{\partial \phi_{b_k n}}$.

- Next, we see a simple way to find such paths
C. Finding Feasible Descent Direction

- Given $\phi$, create graph $R(\phi)$ as follows
  - If $\phi_{ij} > 0$, $F_{ij} < c_{ij}$, then
    - assign weight $\frac{\partial D(\phi)}{\partial \phi_{ij}}$ to $(i, j)$
  - If $\phi_{ij} > 0$, $F_{ij} > 0$, then
    - assign weight $\frac{\partial D(\phi)}{\partial \phi_{ij}}$ to $(j, i)$.

- Note that in $R(\phi)$, a feasible descent direction is
  - Equivalent to a negative cost cycle

- We’ll look for min-cost path in $R(\phi)$
  - The Dijkstra’s algorithm does not work
    - due to negative weights
  - We will use Bellman–Ford algorithm to detect neg-cycles
D. Optimality at Convergence

- Convergent point $\phi^*$
  - Exists because $D(\phi)$ is always decreasing
  - Further, $\phi^*$ is such that no descent direction.
  - This implies, $\phi^*$ is optimal.

- The described algorithm is
  - centralized
    - not implementable
  - However, it’s very instructive for general optimization problems

- Next, we’ll see a distributed algorithm
  - Based on “dual-decomposition” and “subgradient” methods.
  - Similar ideas will be used in the context of congestion-control.
Distributed Route-Opt

• Main method
  ◦ Primal & Dual version
    – convexity implies: solving Primal = solving Dual
  ◦ Interestingly,
    – Dual can be solved in distributed manner
    – using ‘subgradient’ method which is extension to gradient descent algorithm

• The algorithm is very similar in nature to Dijkstra’s algorithm
Primal (P):

$$\min \sum_{(i,\ell) \in \mathcal{L}} D_{i\ell}(F_{i\ell}) \triangleq \sum_{e \in \mathcal{L}} D_e(F_e)$$

subject to

$$C = \begin{bmatrix} F_{i\ell} &=& \sum_k t_i(k)\phi_{i\ell}(k) & \leq c_{i\ell} \\ \phi_{i\ell}(j) &\geq& 0 , & \text{for all } i, j, \ell; \\ \sum_{\ell} \phi_i(j) & = & 1 , & \text{for all } \ell, j. \end{bmatrix}$$

$$O = [t_i(j) = r_i(j) + \sum_{\ell} t_{\ell}(j)\phi_{\ell i}(j) \text{ for all } i, j;]$$

- Reformulation: $x_{ik}(j) = t_i(j)\phi_{ik}(j)$

- Then,

$$C \iff \{ F_{i\ell} = \sum_k x_{i\ell}(k) \leq c_{i\ell} ; \quad x_{i\ell}(k) \geq 0 \}$$

$$O \iff \sum_{k:(ik) \in \mathcal{L}} x_{ik}(j) = r_i(j) + \sum_{\ell:(\ell,i) \in \mathcal{L}} x_{\ell i}(j).$$
Dual of Route-Opt

- **Lagrangian:**
  \[
  L(x; \gamma) = \sum_{e \in \mathcal{E}} D_e(F_e) + \sum_{i,j} \gamma_i(j) \left[ -\sum_k x_{ik}(j) + \sum_\ell x_{\ell i}(j) + r_i(j) \right]
  \]
  \[
  = \sum_{e \in \mathcal{E}} \left[ D_e(F_e) + \sum_{j=1}^n (\gamma_e^+(j) - \gamma_e^-(j))x_e(j) \right] + \gamma^T r
  \]
  where \(e = (e^-, e^+)\)

- **Dual function:**
  \[
  q(\gamma) = \inf_{x \in \mathcal{C}} L(x; \gamma)
  \]
  \[
  = \gamma^T r + \sum_{e \in \mathcal{E}} x_e=(x_e(j));x_e\in\mathcal{C} \left[ D_e(F_e) + \sum_{j=1}^n x_e(j)\nabla \gamma_e(j) \right]
  \]
  where \(\nabla \gamma_e(j) = \gamma_e^+(j) - \gamma_e^-(j)\).

- Thus, \(q(\gamma)\) can be evaluated "locally", since
  \(x_e \in \mathcal{C}\) is “locally” checkable.
Dual of Route-Opt

- **Dual (D):**
  \[
  \max_{\gamma} q(\gamma).
  \]

- Since there is no duality gap (P is convex minimization)
  - \(\gamma^*\) s.t. \(q(\gamma^*) = \max_{\gamma} q(\gamma)\) gives \(x^*(\gamma^*) = L(x^*(\gamma^*), \gamma^*)\).
  \[
  \Rightarrow \text{Sufficient to solve } D
  \]

- In D, \(\gamma\) is "free" variable, hence
  - If we could evaluate \(q(\gamma)\) locally, and
  - Use simple unconstrained optimization procedure
  - We’ll obtain distributed solution

- Question: what unconstrained optimization procedure?
  - Can not used gradient descent as it gradient may not exists
  - Instead, we’ll use subgradient method
Subgradient Method

- Similar to gradient, but used when gradient is non-unique

\[ \max_\gamma q(\gamma) = \min_\gamma -q(\gamma) : \text{standard convex minimization} \]

\[ -q(\gamma) = - \inf_x L(x, \gamma) = -L(x(\gamma), \gamma) . \]

- Then, gradient of \( L(x(\gamma), \gamma) \) is a subgradient for \( (-q(\gamma)) \)

\[ -\nabla L(x(\gamma), \gamma) = \left( - \frac{\partial q(\gamma)}{\partial \gamma_i(j)} \right) \]

where,

\[
\frac{\partial q(\gamma)}{\partial \gamma_i(j)} = r_i(j) \left[ \sum_{e:e^+ = i} x_e(j) - \sum_{e:e^- = i} x_e(j) \right] + \\
\left[ \sum_{e:e^+ = i \text{ or } e^- = i} \frac{\partial D_e(x_e)}{\partial x_e} \left\{ \sum_{j=1}^{n} \frac{\partial x_e(j)}{\partial \gamma_i(j)} \right\} \right] \\
+ \sum_{e:e^+ = i} \sum_{j=1}^{n} \frac{\partial x_e(j)}{\partial \gamma_i(j)} - \sum_{e:e^- = i} \sum_{j=1}^{n} \frac{\partial x_e(j)}{\partial \gamma_i(j)} .
\]
Subgradient Method

- Hence, to compute subgradient of \((-q(\gamma))\), we need
  \[
  \left( \frac{\partial x_i(j)}{\partial \gamma_i(j)} \right)_{i,j} \text{ at } x = x(\gamma) \quad \text{and} \quad \frac{\partial D(x_e)}{\partial x_e} \text{ at } x = x(\gamma)
  \]
  - Again, these are locally computable
  - Hence, subgradient components \(\frac{\partial(-q(\gamma))}{\partial \gamma_i(j)}\) are computable at node \(i\) (using edge variables)

- Subgradient Algorithm

  1. Start with initial \(\gamma^0\). Set \(t = 0\).
  2. Compute \(q(\gamma^t)\).
  3. Compute subgradient of \(-q(\gamma^t) = (G^t_i(j))\).
  4. Update \(\gamma^{t+1} = \gamma^t - \alpha_t G^t\).
  5. Set \(t = t + 1\) and repeat from \#2.

* Here, \(\alpha_t\) is s.t. \(\lim \alpha_t = 0\) but \(\sum \alpha_t = \infty\).
Summary

• Routing
  ○ An essential network-layer task
  ○ Simple flow-based model to describe the setup formally
    – allows us to evaluate performance
  ○ Heuristics
    – simple, distributed and, hence, implemented
  ○ Optimal routing
    – at the final glance, solvable but difficult to implement
    – recent techniques can lead to simple, implementable solutions
    – will it be implemented?
Summary

- Related results
  - Completely distributed primal algorithm
    - Algorithm by Gallager (1976)
  - Asynchronous algorithms
    - Algorithm by Tsitsiklis and Bertsekas (1985)
  - Effect of failure
  - Stability of algorithm (no oscillation)

- Broad impact
  - Led to development of distributed network algorithms
    - similar ideas in congestion control
  - Routing or job assignment tasks in other scenarios can benefit from these methods
References

1. Chapter 5, in book on Data Networks by Bertsekas-Gallager
2. Notes by Stephen Boyd (some posted on the class page)
   A. Link 1: www.stanford.edu/course/ee3920/
      – Subgradient: definition and properties
      – Subgradient algorithm: convergence and correctness
   B. Link 2: www.stanford.edu/course/ee363/ [or Chapters 4 and 5, Convex Optimization by Boyd-Vandenberg]
   C. Notes on Decomposition Method
      – Again, see www.stanford.edu/course/ee3920/, or
      – Chapter 6, Nonlinear Programming by Betsekas
3. Miscellaneous
   – Some notes of current Internet routing practice will be posted
   – An excellent survey paper by Tsitsiklis and Bertsekas (1992) on Asynchronous algorithms