Network Flows for Functions

D. Manjunath
[Joint work with Virag Shah and Bikash Kumar Dey]

IIT Bombay

NCC 2011; 29 Jan 2011
Outline

- The problem setup
- The LP formulations and algorithms
- Extensions and open problems
- Other works
- Summary
The setting

- Terminal (aka sink) nodes want to recover functions of data from distributed sources. Example of functions: max, min, average, etc.
- Application example: Average temperature sensed by many sensors, average/maximum traffic at different parts of a network.
The setting
Computing function: a simple example

\[ f(X,Y,Z) = \max(X,Y,Z) \]
Computing function: a simple example

\[ f(X, Y, Z) = \max(X, Y, Z) \]

- Computing at internal nodes is a natural choice—distributed computation of distributed data.
- We assume: no mixing of data across different realizations.
Computation tree of $\Theta(X_1, X_2, X_3) = X_1 X_2 + X_3$
A single computation tree may serve different functions.

Our techniques depend only on the computation tree and not on the function.

A single function may allow multiple computation trees; we start by assuming a single computation tree and generalise to multiple trees.

\[
\Theta(X_1, X_2, X_3) = X_1 X_2 + X_3
\]

OR

\[(X_1 + X_2)X_3\]

OR

\[\vdots\]
Our setup

- A *single* terminal wants to compute a *single* function of the distributed using a given computation tree. Many generalizations will follow.
Our setup

- A single terminal wants to compute a single function of the distributed using a given computation tree. Many generalizations will follow.

- The network has undirected half-duplex links with a total capacity constraint. Easily applicable to directed networks.

\[ \text{total flow} \leq c(uv) \]
Our setup

- A *single* terminal wants to compute a *single* function of the distributed using a given computation tree. Many generalizations will follow.

- The network has undirected half-duplex links with a total capacity constraint. Easily applicable to directed networks.

  \[ \text{total flow} \leq c(uv) \]

- Objective: to find the *maximum computation rate* per use of the network and to find an *optimum computation and communication scheme*. 
Embedding: Illustration for $\Theta = X_1 X_2 + X_3$

$\mathcal{N}$ and $\mathcal{G}$ and two possible embeddings.
Definition

A path in $\mathcal{N}$ is a sequence of nodes $v_1, v_2, \cdots, v_l; \ l \geq 1$ s.t. $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l - 1$.

Let $\mathcal{P}$ denote the set of paths in $\mathcal{N}$.
Definition

A path in $\mathcal{N}$ is a sequence of nodes $v_1, v_2, \cdots, v_l; \ l \geq 1$ s.t. $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l - 1$.

Let $\mathcal{P}$ denote the set of paths in $\mathcal{N}$.

Define $\Phi_\uparrow(\theta) \triangleq \{ \eta \in \Gamma | \text{head}(\eta) = \text{tail}(\theta) \}$ and $\Phi_\downarrow(\theta) \triangleq \{ \eta \in \Gamma | \text{tail}(\eta) = \text{head}(\theta) \}$. 
Definition

A path in $\mathcal{N}$ is a sequence of nodes $v_1, v_2, \ldots, v_l; \ l \geq 1$ s.t. $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l - 1$.

Let $\mathcal{P}$ denote the set of paths in $\mathcal{N}$.

Define $\Phi^\uparrow(\theta) \triangleq \{ \eta \in \Gamma | \text{head}(\eta) = \text{tail}(\theta) \}$ and $\Phi^\downarrow(\theta) \triangleq \{ \eta \in \Gamma | \text{tail}(\eta) = \text{head}(\theta) \}$.

The edges of $\mathcal{G}$ are ordered in a topological order.

**Definition:** An embedding of $\mathcal{G}$ into $\mathcal{N}$ is a map $B : \Gamma \rightarrow \mathcal{P}$ such that

1. $\text{start}(B(\theta_l)) = s_l$ for $l = 1, 2, \ldots, \kappa$
2. $\text{end}(B(\eta)) = \text{start}(B(\theta))$ if $\eta \in \Phi^\uparrow(\theta)$
3. $\text{end}(B(\theta_{|\Gamma|})) = t$. 
Embedding-Edge LP

- What is the best time-sharing between the different embeddings?

**Embedding-Edge LP**: Maximize \( \lambda = \sum_{B \in \mathcal{B}} x(B) \) subject to

1. Capacity constraints

\[
\sum_{B \in \mathcal{B}} r_B(e) x(B) \leq c(e), \quad \forall e \in E
\]

(1)

2. Non-negativity constraints

\[
x(B) \geq 0, \quad \forall B
\]

(2)

\( r_B(e) \) = \# of times that network edge \( e \) of \( \mathcal{N} \) is used in embedding \( B \).
Embedding-Edge LP

- A protocol can be designed from any feasible solution of the Embedding-Edge LP to achieve a computation rate of $\lambda$. 
Embedding-Edge LP

- A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of $\lambda$.

- $|\mathcal{B}|$ is exponential in $|V|$. So this LP has exponential complexity.
• A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of $\lambda$.

• $|\mathcal{B}|$ is exponential in $|V|$. So this LP has exponential complexity.

• We need to seek simpler solutions
  • We can identify each edge of $\mathcal{G}$ to be a flow.
Embedding-Edge LP

- A protocol can be designed from any feasible solution of the *Embedding-Edge LP* to achieve a computation rate of $\lambda$.

- $|B|$ is exponential in $|V|$. So this LP has exponential complexity.

- We need to seek simpler solutions
  - We can identify each edge of $\mathcal{G}$ to be a flow.
  - We can thus explore an efficient *Node-Arc LP* based on “flow conservation.”
Flow-conservation

- A complication: Flow can be destroyed or generated at internal nodes.
Flow-conservation

- A complication: Flow can be destroyed or generated at internal nodes.

- A destroyed flow of type $\theta_1$ or $\theta_2$ generates a flow of type $\eta$ of the same volume.
Flow-conservation

- A complication: Flow can be destroyed or generated at internal nodes.

- A destroyed flow of type $\theta_1$ or $\theta_2$ generates a flow of type $\eta$ of the same volume.

- A flow generated at a node is assumed to flow on a virtual self-loop at that node.

- The flow in the self-loop contributes to the incoming flow, but not to the outgoing flow.
Flow-conservation

\[
\begin{align*}
X_1 & \quad X_2 & \quad X_3 \\
s_1 & \quad s_2 & \quad s_3 \\
X_1 & \quad X_2 & \quad X_3 \\
t & \quad t & \quad t
\end{align*}
\]
Flow-conservation

\[ f^{x_1} = 1.5 \]
\[ f^{x_2} = 1.5 \]
\[ f^{x_3} = 1.5 \]

\[ f^{x_1x_2} = 1.5 \]
\[ f^{x_1x_2+x_3} = 1 \]
\[ f^{x_1x_2} = 0.5 \]
\[ f^{x_1x_2+x_3} = 0.5 \]

\[ f^{x_3} = 1 \]
\[ f^{x_3} = 1.5 \]
\[ f^{x_3} = 0.5 \]

\[ f^{x_1x_2} = 1.5 \]
\[ f^{x_2} = 1.5 \]
\[ f^{x_3} = 1.5 \]
**Node-Arc LP**

*Node-Arc LP:* Maximize $\lambda$ subject to following constraints. For any node $v \in V$

1. Functional conservation of flows:

$$f_{\eta v}^\theta + \sum_{u \in N(v)} f_{v u}^\theta - \sum_{u \in N'(v)} f_{u v}^\theta = 0,$$

\[ \forall \theta \in \Gamma \setminus \{\theta|_{\Gamma}\} \text{ and } \forall \eta \in \Phi_{\downarrow}(\theta). \]

2. Conservation and termination of $\theta|_{\Gamma}$:

$$\sum_{u \in N(v)} f_{v u}^{\theta|_{\Gamma}} - \sum_{u \in N'(v)} f_{u v}^{\theta|_{\Gamma}} = \begin{cases} -\lambda & v = t \\ 0 & \text{otherwise} \end{cases}$$
Node-Arc LP

3. Generation of $\theta_l \ \forall l \in \{1, 2, \ldots, \kappa\}$:

$$f^{\theta_l}_{vv} = \begin{cases} 
\lambda & v = s_l \\
0 & \text{otherwise}
\end{cases}$$

4. Capacity constraints

$$\sum_{\theta \in \Gamma} \left( f^{\theta}_{uv} + f^{\theta}_{vu} \right) \leq c(uv), \ \forall uv \in E.$$ 

5. Non-negativity constraints

$$f^{\theta}_{uv} \geq 0, \ \forall uv \in E \ \text{and} \ \forall \theta \in \Gamma$$

$$f^{\theta}_{uu} \geq 0, \ \forall u \in V \ \text{and} \ \forall \theta \in \Gamma$$

$$\lambda \geq 0.$$
Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is a similar to LP formulations in multi-commodity flow.
Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is similar to LP formulations in multi-commodity flow.
- An algorithm $Extract$-$Embedding$ converts a solution of Node-Arc LP to a solution of Embedding-Edge LP.
Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is similar to LP formulations in multi-commodity flow.
- An algorithm $Extract$-$Embedding$ converts a solution of Node-Arc LP to a solution of Embedding-Edge LP.
- In each iteration, $Extract$-$Embedding$ extracts a non-zero flow on an embedding or a nonzero redundant flow on a cycle.
Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is similar to LP formulations in multi-commodity flow.
- An algorithm *Extract-Embedding* converts a solution of Node-Arc LP to a solution of Embedding-Edge LP.
- In each iteration, *Extract-Embedding* extracts a non-zero flow on an embedding or a nonzero redundant flow on a cycle.
- At least one flow $f_{uv}^\theta$ is completely removed in each iteration of Extract-Embedding.
Node-Arc LP $\rightarrow$ Embedding-Edge LP

- This is similar to LP formulations in multi-commodity flow.
- An algorithm Extract-Embedding converts a solution of Node-Arc LP to a solution of Embedding-Edge LP.
- In each iteration, Extract-Embedding extracts’ a non-zero flow on an embedding or a nonzero redundant flow on a cycle.
- At least one flow $f_{uv}^\theta$ is completely removed in each iteration of Extract-Embedding.
- The algorithm has the overall complexity $O(\kappa^2 |E|^2)$.
- The LP has $O(\kappa|E|)$ number of variables, $O(\kappa|E|)$ number of non-negativity constraints, and $O(\kappa|V| + |E|)$ number of other constraints.
Toward an efficient $\epsilon$-approximate solution

- For multi-commodity flow, and more general packing LPs, Garg and Konemann [1998] gave a primal-dual algorithm to compute a solution which achieves at least $(1 - \epsilon)$ fraction of the optimal rate.
- Our *Embedding-Edge LP* is also such a ‘packing LP’.
- So, Garg-Konemann algorithm can be used for our problem.
- The algorithm uses an oracle subroutine that solves a ‘dual’ problem.
Toward an efficient \( \varepsilon \)-approximate solution

Let \( l(e) \) be the weight of edge \( e \). Define the weight of an embedding \( B \) as

\[
    w_L(B) = \sum_{e \in B} r_B(e)l(e).
\]

- The oracle subroutine \( \text{OptimalEmbedding}(L) \) finds an embedding with minimum weight for a given set \( L \) of edge weights.
Toward an efficient $\epsilon$-approximate solution

Let $l(e)$ be the weight of edge $e$. Define the weight of an embedding $B$ as

$$w_L(B) = \sum_{e \in B} r_B(e)l(e).$$

- The oracle subroutine $OptimalEmbedding(L)$ finds an embedding with minimum weight for a given set $L$ of edge weights.
- We later give an efficient algorithm for doing this.
An efficient $\epsilon$-approximate solution

Algorithm: Algorithm for finding approximately optimal $x$ and $\lambda$

Input: Network graph $\mathcal{N} = (V, E)$, capacities $c(e)$, set of source nodes $S$, terminal node $t$, computation tree $\mathcal{G} = (\Omega, \Gamma)$, the desired accuracy $\epsilon$

Output: Primal solution $\{x(B), B \in \mathcal{B}\}$

Initialize $l(e) := \delta / c(e), \forall e \in E, x(B) := 0, \forall B \in \mathcal{B}$

while $D(l) < 1$ do
  \% $D(l) = \sum c(e)l(e)$
  $B^* := \text{OptimalEmbedding}(L)$;
  $e^* := \text{edge in } B^* \text{ with smallest } c(e)/r_{B^*}(e)$;
  $x(B^*) := x(B^*) + c(e^*)/r_{B^*}(e^*)$;
  $l(e) := l(e)(1 + \epsilon \frac{c(e^*)/r_{B^*}(e^*)}{c(e)/r_{B^*}(e)})$, $\forall e \in B^*$;

end

$x(B) := x(B)/\log_{1+\epsilon} \frac{1+\epsilon}{\delta}, \forall B$;
**OptimalEmbedding(\(L\))**: overview

- For each edge \(\theta_i\), starting from \(\theta_1\), it finds a way to compute \(\theta_i\) at each network node at the minimum cost possible.

- It keeps track of that minimum cost and also the ‘predecessor’ node from where it receives \(\theta_i\).

- If \(\theta_i\) is computed at that node itself then the predecessor node is itself.
**OptimalEmbedding**($L$)

- Computing $\theta_i$ for $i \in \{1, 2, \ldots, \kappa\}$ at the minimum cost at a node $u$ is equivalent to finding the shortest path to $u$ from $s_i$. We do this by using Dijkstra’s algorithm.

- For any other $i$, the node $u$ can either compute $\theta_i$ from $\Phi_{\uparrow}(\theta_i)$ or receive it from one of its neighbors.

- To take this into account, unlike Dijkstra’s algorithm, we initialize the cost of computing $\theta_i$ with the cost of computing $\Phi_{\uparrow}(\theta_i)$ at the same node. The rest is similar to Dijkstra’s algorithm.

- Finally the predecessors are backtracked from $t$ to find the optimal embedding.
Complexity

- Overall complexity of $\text{OptimalEmbedding}(L)$:
  $O(\kappa(|E| + |V| \log |V|))$

- The number of iterations in the primal-dual algorithm is of the order $O(\epsilon^{-1}|E| \log_{1+\epsilon} |E|)$.

- Thus the overall complexity of the primal-dual algorithm is $O \left( \epsilon^{-1} \kappa |E| (|E| + |V| \log |V|) \log_{1+\epsilon} |E| \right)$. 
Extensions and Open Problems

**Extensions**
- Multiple trees for the same function.
- Multiple terminals and functions of distinct sources.
- Computing with a specified precision.
- Consider energy limited sensors.

**Open problems**
- An immediate open problem: The computation graph $\mathcal{G}$ is a DAG and not a tree.
In perspective: Other setups for function networks

- Wired/wireless networks: Graph or hypergraph
  \[ u \rightarrow v \]
  \[ v \rightarrow u \]
- Directed or undirected links
  \[ u \rightarrow v \]
  \[ u \rightarrow v \]
  \[ \text{flow} \leq c(uv) \]
  \[ \text{total flow} \leq c(uv) \]
- block computation/coding vs. bit-wise computation
- zero-error recovery vs. small-error recovery
- correlated vs. independent sources
- single terminal vs. multiple terminals
- same vs. different functions at different terminals
- fixed vs. random networks
Typical objectives

- For any given link capacities, what is the maximum rate (or rate-region for multiple terminals) that is achievable? More generally, rate-distortion trade-off?
- Asymptotic scaling laws for required communication complexity per node
- Efficient encoding/decoding
Summary of other views

- In information theory: Small networks, correlated sources with the objective of finding achievable rate-region and rate-distortion.
- Scaling laws for randomly deployed networks: Does not consider a fixed network.
- Network coding: internal nodes are allowed to mix received data to construct outgoing data even for communication.
The big picture
The big picture

- Computer Science
- Information Theory
- Networking
- Network coding
- Distributed function computation
- Flow for functions
- multi-commodity flow
The End
The End

Thank you
The End

Thank you

Questions?