Network Flows for Functions

Virag Shah Bikash Kumar Dey D. Manjunath

***Abstract*—We consider in-network computation of an arbitrary function over an arbitrary communication network. A network with capacity constraints on the links is given. Some nodes in the network generate data, e.g., sensor nodes in a sensor network. An arbitrary function of this distributed data is to be obtained at a terminal node. The structure of the function is described by a given computation schema, which in turn is represented by a directed tree. We define a new notion of conservation of flow suitable in this setup and design computing and communicating schemes to obtain the function at the terminal at the maximum rate. For this, we formulate linear programs to determine network flows that maximize the computation rate. Our approach introduces the network flow techniques to the distributed function computation setup where such a scope was hitherto unsuspected due to the lack of traditional conservation of flow.**

I. Introduction

Motivated by sensor network applications, there has been significant interest in computing functions of distributed data inside the network. The interest of the network is to obtain a function, say Θ, of the source symbols at a terminal node. The nodes in the network can perform computation and thereby participate in the computation of Θ. In this setting, we assume that the variables form a time sequence and that they can be generated at any rate; equivalently, an infinite sequence is readily available. Thus, it is desired to compute Θ at the maximum possible rate. In this paper we introduce novel network flow techniques to design a computation and communication scheme that maximizes the rate at which Θ is computed. Though network flow techniques have been used widely to study multiple unicast [1], [2] problems, also known as multi-commodity problems, our work develops such techniques for the first time for function computation.

A major body of work on in-network computation exists on the asymptotic analysis of the number of transmissions needed to compute specific functions in noisy broadcast networks (e.g., [3] and follow-up works by various authors,) and more recently, in networks represented by random geometric graph models, e.g., [4], [5]. Another class of work, in information theory, considers simplistic fixed networks with small number of correlated sources, e.g. [6], [7]. There has been some recent work in the network coding literature on distributed function computation over larger and more complex networks with independent sources. However, designing optimal coding schemes and finding capacity is a difficult problem except for very special functions or networks [8], [9].

In this paper we make a significant departure from all the above in the problem setup as well as in the techniques used. We consider arbitrary functions of the distributed data, and an arbitrary network with directed or undirected edges with capacity constraints. To illustrate the essence of our setup, consider the function $\Theta(X_1, X_2, X_3) = X_1X_2 + X_3$ of three variables generated at three sources s_1, s_2, and s_3 respectively. A terminal node t needs to obtain $\Theta(X_1, X_2, X_3)$. We assume that all the three data symbols are from the same alphabet $\mathcal{A}$. The computation of the function can be broken into two parts—first computing X_1X_2, and then adding X_3. These two operations can be done at different nodes in the network in the above order. Such a sequence of operations to compute the function is called a *computation schema*, and is represented by a *computation tree* as shown in Fig. 1(b) for the above Θ.

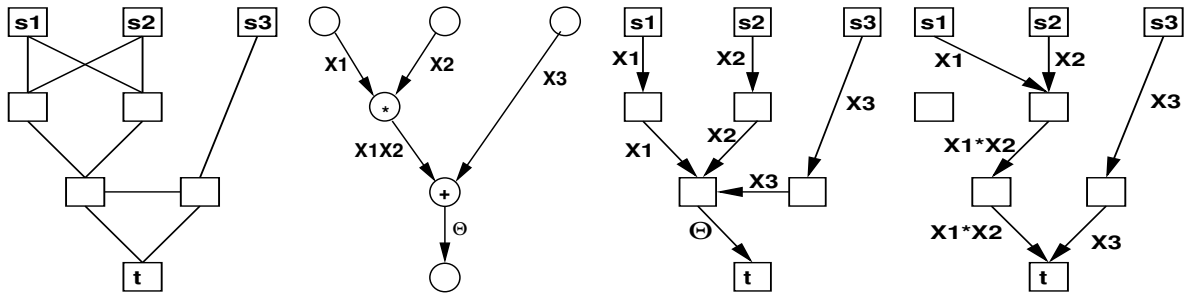
Now consider computing $\Theta(X_1, X_2, X_3)$ in the network shown in Fig. 1(a) where each edge has unit capacity. Two such ways of computing this function are shown in Figs. 1(c) and 1(d). These are called 'embeddings', defined formally in Sec. II. Our aim is to find the best time-sharing between these various embeddings to achieve the maximum number of computations per use of the network.

A given function may allow computation using different computation trees. To find the optimum solution, one has to find the best timesharing between all the embeddings of all such computation trees in the given network. For most of this paper, we consider a single given computation tree to illustrate our techniques. We then extend our techniques (in Sec. III-B) for the cases when multiple computation trees are given for the function. We note that the 'star' tree is a computation tree for any function, and it corresponds to communicating all the data symbols to a single node for computation of the function and subsequently forwarding to the terminal node. Note that computation of a function even on an isolated computer needs a sequence of operations, and the knowledge of one or more computations trees is a basic and natural requirement. For any given set of computation trees for a function, our approach gives the optimum computation-rate under the use of the corresponding computation-sequences. We do not address the question of how to find computation trees for Θ in this paper.

Organization and contribution: We describe the model in Sec. II. Section III presents the main contributions of this paper. We formulate a linear program, *Embedding-Edge-LP,* that defines the problem of optimally allocating flows on the embeddings to achieve the maximum computation rate. We then introduce a notion of flow conservation, and present an LP, *Node-Arc-LP,*, that can be solved in polynomial time. We then present an efficient algorithm, *Algorithm 1,* that converts the edge-flows obtained from *Node-Arc-LP* into a flow allocation on the embeddings.

In an extended version of the paper [12], we also develop

V. Shah is with ECE Dept., Univ. of Texas at Austin. B. K. Dey and D. Manjunath are with EE Deptt., IIT Bombay. Research supported by a project from DST, Govt. of India and done at Bharti Centre for Communication in IIT Bombay. Emails: virag4u@gmail.com,{bikash,dmanju}@ee.iitb.ac.in



(a) (b) (c) (d)

Fig. 1. Computing $\Theta = X_1X_2 + X_3$ over a network. (a) A network to compute $\Theta = X_1X_2 + X_3$. (b) A schema represented by a computation tree for computing Θ (c) A possible embedding that computes at Θ at unit rate. (d) An alternative embedding.

a primal-dual algorithm that is faster than a solution via *Node-Arc-LP* and *Algorithm 1*, and obtains an approximately optimal (within $(1-\epsilon)$ fraction) solution. As a subroutine for this algorithm, we develop an efficient algorithm for finding a minimum-weight embedding in a network with weighted links, which is of independent interest. We also show that our techniques can be extended to interesting practical scenarios of multiple terminals wanting different functions, computation with a given desired precision, and to networks with energy constraints at nodes.

II. The model and the notation

The communication network is an undirected, simple, connected graph $\mathcal{N} = (V, E)$ where V is a set of n nodes and E is a set of m undirected edges. Each edge $uv \in E$ represents a half duplex link with a total non-negative capacity $c(uv)$. In the network, $S = \{s_1, s_2, \ldots, s_\kappa\} \subset V$ is the set of κ source nodes. Source s_i has an infinite sequence of data values $\{X_i(k)\}_{k\geq 0}$ where $X_i(k)$ belongs to a finite alphabet $\mathcal{A}$. X_i is used to denote a representative element of the sequence. Let $X \stackrel{\triangle}{=} [X_1, \ldots X_\kappa]$. The link capacities are expressed in $|\mathcal{A}|$-ary unit. Without loss of generality, we assume that each source node in the network generates exactly one data sequence. We also assume that there is only one terminal node.

A given function $\Theta : \mathcal{A}^\kappa \to \mathcal{A}$ of X needs to be obtained at the terminal node t for each k at the best possible rate. A computation schema for Θ is given and represented by a directed tree $\mathcal{G} = (\Omega, \Gamma)$ where Ω is the set of nodes and Γ is the set of edges. The elements of Ω are labeled $\mu_1, \mu_2, \ldots, \mu_{|\Omega|}$ where $\mu_1, \mu_2, \ldots, \mu_\kappa$ are the source nodes, $\mu_{|\Omega|}$ is the terminal node that obtains Θ and the rest are computing nodes that compute different functions of X. Further, the nodes in Ω are labeled according to a topological order such that for $i > j$ there is no directed path in $\mathcal{G}$ from μ_i to μ_j. The source nodes have in-degree zero and out-degree one and the terminal node has in-degree one and out-degree zero. All other nodes have in-degree greater than one and out-degree exactly one. Similarly, the elements of Γ are labeled $\theta_1, \theta_2, \ldots, \theta_{|\Gamma|}$ with $\theta_1, \theta_2, \ldots, \theta_\kappa$ being the outgoing edges from $\mu_1, \mu_2, \ldots, \mu_\kappa$ respectively, and $\theta_{|\Gamma|}$ being the incoming edge into $\mu_{|\Omega|}$. The remaining edges are labeled according to a topological order, i.e., for any $i < j$, there is no path from the head node of j to the tail node of i.

For any edge $\theta \in \Gamma$, let $tail(\theta)$ and $head(\theta)$ represent, respectively, the tail and the head nodes of the edge θ. Let $\Phi_\uparrow(\theta)$ and $\Phi_\downarrow(\theta)$ denote, respectively, the predecessors and the successors of θ, i.e., $\Phi_\uparrow(\theta) \stackrel{\triangle}{=} \{\eta \in \Gamma | head(\eta) = tail(\theta)\}$ and $\Phi_\downarrow(\theta) \stackrel{\triangle}{=} \{\eta \in \Gamma | tail(\eta) = head(\theta)\}$. We assume that each edge θ of $\mathcal{G}$ represents a distinct function of X that can be computed from the functions corresponding to the edges in $\Phi_\uparrow(\theta)$. Further, each function takes values from the same alphabet $\mathcal{A}$. (We remark that this is not unreasonable even when all the computations are over real numbers because computations are performed using a fixed precision.)

Let $N(v) \stackrel{\triangle}{=} \{u \in V | uv \in E\}$ denote the set of neighbors of a node $v \in V$ and $N'(v) = N(v) \cup \{v\}$. A sequence of nodes $v_1, v_2, \cdots, v_l$, $l \geq 1$, is called a path if $v_iv_{i+1} \in E$ for $i = 1, 2, \ldots, l-1$. The set of all paths in $\mathcal{N}$ is denoted by $\mathcal{P}$. With abuse of notation, for such a path P, we will say $v_i \in P$ and also $v_iv_{i+1} \in P$. The nodes v_1 and v_l are called respectively the start node and the end node of P, and are denoted as start(P) and end(P).

As discussed in Sec. I, a function with a given computation tree can be computed along any "embedding" of the tree in the network as shown in Fig. 1. We are now ready to formally define an embedding of a computation tree.

Definition: An embedding is a mapping $B : \Gamma \to \mathcal{P}$ such that

1) start$(B(\theta_l)) = s_l$ for $l = 1, 2, \ldots, \kappa$
2) end$(B(\eta)) =$ start$(B(\theta))$ if $\eta \in \Phi_\uparrow(\theta)$
3) end$(B(\theta_{|\Gamma|})) = t$.

We denote the set of embeddings of $\mathcal{G}$ in $\mathcal{N}$ by $\mathcal{B}$. Our aim is to determine the flow on each of these embeddings so as to maximize the total flow. An edge in the network may carry different functions of the source data in an embedding. We thus define the number of times an edge $e \in E$ is used in an embedding B as $r_B(e) = |\{\theta \in \Gamma | e \text{ is a part of } B(\theta)\}|$. An edge may also be used to carry flows on different embeddings. Therefore in an assignment of flows on different embeddings, i.e., in a particular timesharing scheme, the edge may carry multiple types of data (i.e., different functions of X) of different amounts. Also note that, if start$(B(\theta_i)) =$ end$(B(\theta_i))$, i.e., if $B(\theta_i)$ consists of a single node, then in that embedding the data θ_i is generated as well as used (i.e., not forwarded to another node) in that node.

III. Linear programs and algorithms

In this section, we present our main contributions. We first give a basic linear program, the *Embedding-Edge LP*, which characterizes our problem.

As discussed in Sec. I and Sec. II, the function for a particular sample of the data can be computed over the network using any embedding of the computation tree in the network. For any embedding $B \in \mathcal{B}$, let $x(B)$ denote the average number of function symbols computed using the embedding B per use of the network. We present below a linear program to maximize $\lambda := \sum_{B\in\mathcal{B}} x(B)$. Recall that $r_B(e)$ represents the number of times the edge e is used in the embedding B.

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

1. Capacity constraints:

$$\sum_{B\in\mathcal{B}} r_B(e)x(B) \leq c(e),\ \forall e \in E \tag{1}$$

2. Non-negativity constraints:

$$x(B) \geq 0,\ \forall B \tag{2}$$

This LP finds an optimal fractional packing of the embeddings of $\mathcal{G}$ into $\mathcal{N}$.

In multi-commodity flow problems, a solution of the so called *Path-edge LP* readily gives a way of achieving the corresponding rates. However, since in our problem, the data is to be mixed according to different embeddings for different realizations of data, one needs to carefully devise a protocol to schedule the computation and communication at the nodes and edges in such a way that data from different realizations are not mixed. Such a protocol is presented in [12].

A. The Node-Arc LP

Note that the cardinality of $\mathcal{B}$ can be exponential in $|V|$. Hence the complexity of *Embedding-Edge LP* is exponential in the network parameters if some structure of the problem is not used. In the multi-commodity flow literature, another LP formulation, called the *Node-Arc LP*, based on the flow conservation principle is well-known and can be solved in polynomial time. In the following, we point out a flow conservation that holds in our setup and present an LP for our problem based on this.

We assume that each node in the network has a virtual self-loop of infinite capacity. The data flowing in the self-loop represents the data generated at that node. This may be the source data generated at the sources or the intermediate or final values computed at the node. For example, if a node computes X_1X_2 from X_1 and X_2 it receives, and then computes $X_1X_2 + X_3$ by using the computed X_1X_2 and received X_3, then both X_1X_2 and $X_1X_2+X_3$ will be assumed to flow in its self-loop. Example of the flows on the edges and the self-loops corresponding to a particular flow assignment on two embeddings is shown in Fig. 2.

The variables in our *Node-Arc LP* are $\left\{f^\theta_{uv}, f^\theta_{vu} | uv \in E, \theta \in \Gamma\right\} \cup \left\{f^\theta_{uu} | u \in V, \theta \in \Gamma\right\} \cup \{\lambda\}$.

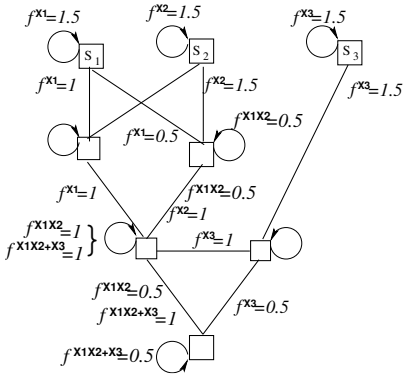


Fig. 2. The aggregate edge-flow values for a flow of 0.5 on the embedding in Fig. 1(d) and a flow of 1 on the embedding in Fig. 1(c).

where, f^θ_{uv} represents the flow of type $\theta \in \Gamma$ flowing through the edge $uv \in E$ from u to v, f^θ_{uu} denotes the flow of type θ flowing in the self-loop at u and λ represents the total rate of the function computation.

Node-Arc LP consists of capacity constraints on the edges of $\mathcal{N}$, a flow-conservation rule on the nodes of $\mathcal{N}$, and non-negativity constraints on the flows f^θ_{uv}. *The flow conservation* rule is based on the fact that an intermediate node in $\mathcal{N}$ can, apart from forwarding the flows it receives, generate a flow of type θ on its self-loop by terminating the same amount of incoming flows of type $\eta \in \Phi_\uparrow(\theta)$. Each source node s_l, in addition, generates λ amount of flow of type θ_l. Similarly, the terminal node t terminates λ amount of flow of type $\theta_{|\Gamma|}$. The *Node-Arc LP* is as follows. Recall that $N'(v)$ denotes the set of the neighbors of v and itself.

Node-Arc LP: Maximize λ subject to following constraints for each node $v \in V$.

1. Functional conservation of flows:

$$f^\eta_{vv} + \sum_{u\in N(v)} f^\theta_{vu} - \sum_{u\in N'(v)} f^\theta_{uv} = 0, \quad \forall \theta \in \Gamma \setminus \{\theta_{|\Gamma|}\} \text{ and } \forall \eta \in \Phi_\downarrow(\theta). \tag{3}$$

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

$$\sum_{u\in N(v)} f^{\theta_{|\Gamma|}}_{vu} - \sum_{u\in N'(v)} f^{\theta_{|\Gamma|}}_{uv} = \begin{cases} -\lambda & v = t \\ 0. & \text{otherwise} \end{cases} \tag{4}$$

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

$$f^{\theta_l}_{vv} = \begin{cases} \lambda & v = s_l \\ 0. & \text{otherwise} \end{cases} \tag{5}$$

4. Capacity constraints (undirected duplex links):

$$\sum_{\theta\in\Gamma} \left(f^\theta_{uv} + f^\theta_{vu}\right) \leq c(uv),\ \forall uv \in E. \tag{6}$$

5. Non-negativity constraints:

$$f^\theta_{uv} \geq 0, \forall uv \in E \text{ and } \forall \theta \in \Gamma \tag{7}$$

$$f^\theta_{uu} \geq 0, \forall u \in V \text{ and } \forall \theta \in \Gamma. \tag{8}$$

Complexity: This LP has $O(\kappa m)$ number of variables, $O(\kappa m)$ number of non-negativity constraints (one for each variable), and $O(\kappa n + m)$ number of other constraints. Hence it can be solved in polynomial time.

The above LP gives a set of flow values on each link. Note that, unlike multi-commodity flow problem, the solution to Node-Arc LP does not readily describe a practical communication and computation protocol over the network. In multicommodity flow problem, if a node forwards fractions of an incoming flow to two different links, it can do so by arbitrarily choosing which data goes to which link. However, in function computation setup, a node cannot do this arbitrarily, only data of the same realization can be mixed at network nodes as per the computation schema. We present an algorithm, Algorithm 1, which, from any feasible solution of this LP, obtains a corresponding feasible solution for the *Embedding-Edge LP* that achieves the same λ.

Algorithm 1: In each iteration of the *while* loop (lines 2-33) we find an embedding with a non-zero flow and remove the corresponding edge-flows to obtain another feasible solution with a reduced rate. For this, we start by finding a mapping of $\theta_{|\Gamma|}$, viz. $B(\theta_{|\Gamma|})$. Its last node is t. If $f_{tt}^{\theta_{|\Gamma|}} > 0$, then we assign $B(\theta_{|\Gamma|}) = t$. Else, we seek an edge vt that carries positive flow of type $f^{\theta_{|\Gamma|}}$. We continue this search backwards till we find a node u for which $f_{uu}^{\theta_{|\Gamma|}} > 0$, and assign the explored path $u \cdots t$ to $B(\theta_{|\Gamma|})$. We now repeat (*for* loop in line 6) this process for all $\theta \in \Phi_{\uparrow}(\theta_{|\Gamma|})$ to find $B(\theta)$ ending at u. When the search for $B(\theta_i), \forall \theta_i \in \Gamma$ is completed, we have successfully found an embedding carrying a positive flow. z' and $z(.)$ keep track of the maximum flow the embedding can carry, which is equal to the minimum of the flows in the edges of the embedding. While exploring nodes to find $B(\theta_i)$, the *If* block starting in line 10 checks for presence of a redundant cycle of flow of type θ_i. If such a cycle is found, it is removed from the explored path and the redundant flow is removed from the cycle.

Proof of correctness of Algorithm 1: The proof of the following statements ensures the correctness of the algorithm.

1) In line 9, such a u exists.
2) If a cycle of redundant flow is found and removed in lines 10-15, then the remaining flows still satisfy the constraints in the LP with total flow (λ) reduced by y.
3) At the end of each iteration of the *while* loop (lines 2-33), the remaining flows still satisfy the constraints in the LP with λ replaced by $\lambda - \lambda'$.
4) The algorithm terminates in finite time.

We now outline a proof of each of these statements. We prove the statements 1)–4) for each iteration of the loops while assuming that all the above claims are true in all the previous iterations of the *while* and *for* loops.

Proof of 1: The current values of the flows satisfy all the constraints in the *Node-Arc LP* with λ replaced by $\lambda - \lambda'$. The algorithm ensures that in this step, the total outgoing flow $\sum_{u \in N(v)} f_{vu}^{\theta} \geq z(v) > 0$. So, by constraints (3) and (4), the total of incoming and generated flows $\sum_{u \in N'(v)} f_{uv}^{\theta} > 0$.

Algorithm 1: Finding equivalent solution of the *Embedding-Edge LP* from a feasible solution of the *Node-Arc LP*.

input : Network graph $\mathcal{N} = (V, E)$, capacities $c(e)$, set of source nodes S, terminal node t, computation tree $\mathcal{G} = (\Omega, \Gamma)$, and a feasible solution to its *Node-Arc LP* that consists of the values of λ, $f_{uv}^{\theta}\ \forall \theta \in \Gamma, \forall uv \in E$, and $f_{uu}^{\theta}\ \forall \theta \in \Gamma,\ \forall u \in V$.
output: Solution $\{x(B) | B \in \mathcal{B}\}$ to the *Embedding-Edge LP* with $\sum_{B \in \mathcal{B}} x(B) = \lambda$.

1 Initialize $\lambda' = 0$
2 **while** $\lambda' \neq \lambda$ **do**
3 $z' := \lambda$;
4 $B(\theta_{|\Gamma|}) := t$;
5 $B(\theta_i) = \emptyset$ for $i = 1$ to $|\Gamma| - 1$;
6 **for** $i :=$ $|\Gamma|$ **to** 1 **do**
7 $v :=$ end$(B(\theta_i))$;
8 $z(v) = z'$;
9 $u :=$ an element in $N'(v)$ such that $f_{uv}^{\theta_i} > 0$;
10 **if** $u \neq v$ *and* $u \in B(\theta_i)$ **then**
11 Let P be the path in $B(\theta_i)$ upto the first appearance of u in it.;
12 Delete P from $B(\theta_i)$. ;
13 $y := \min_{u'v' \in \{uv\} \cup P} \left(f_{u'v'}^{\theta_i} \right)$;
14 $f_{u'v'}^{\theta} := f_{u'v'}^{\theta} - y\ \forall u'v' \in \{uv\} \cup P$
15 **end**
16 **else**
17 $z(u) := \min\left(z(v), f_{uv}^{\theta_i}\right)$;
18 **end**
19 **if** $u \neq v$ **then**
20 Prefix u in $B(\theta_i)$;
21 $v := u$;
22 Jump to line 9 ;
23 **end**
24 **else**
25 $B(\eta) := u, \forall \eta \in \Phi_{\uparrow}(\theta_i)$;
26 $z' = z(u)$;
27 **end**
28 **end**
29 $x(B) := z'$;
30 $\lambda' := \lambda' + x(B)$;
31 $f_{u'v'}^{\theta} := f_{u'v'}^{\theta} - x(B)\ \forall \theta \in \Gamma$ and $\forall u'v' \in B(\theta)$;
32 $f_{v'v'}^{\theta} := f_{v'v'}^{\theta} - x(B)\ \forall \theta \in \Gamma$ and $v' =$ start$(B(\theta))$;
33 **end**

Hence the statement follows.

Proof of 2: We will prove that a cyclic flow on a cycle $v_1, v_2, \cdots, v_l, v_1$ satisfies all the constraints in the *Node-Arc LP* with $\lambda = 0$. Then clearly after subtracting this flow from the edges of the cycle, the remaining flows in the network will still satisfy the constraints with the same λ as before. For a cyclic flow of type θ of volume y, the flow values are $f_{v_i v_{i+1}}^{\theta} = y$ for $i = 1, 2, \cdots, l-1$, $f_{v_l v_1}^{\theta} = y$, and all other flow values

are equal to 0. So, for any node, any nonzero incoming flow is 'compensated' by the same amount of outgoing flow of the same type. All flow values in the self-loops are also 0. So clearly these flows satisfy the constraints in the LP with $\lambda = 0$. This completes the proof.

Proof of 3: Again, we will prove that the removed $x(B)$ amount of flows on the edges of an embedding and on the self-loops themselves satisfy the constraints in the LP with $\lambda = x(B)$. Then the remaining flows will also satisfy the constraints with λ replaced by $\lambda - x(B)$. The subtracted flow values are $f^{\theta}_{uv} = x(B)$ for $uv \in B(\theta)$, $f^{\theta}_{uu} = x(B)$ for $u = \text{start}(B(\theta))$, and all other flow values 0. We can verify that these flows satisfy the constraints in the *Node-Arc LP*.

Proof of 4: The *Node-Arc LP* has $O(m|\Gamma|)$ number of variables f^{θ}_{uv} and f^{θ}_{uu}. Each deletion of flows through a cycle, or through an embedding, makes at least one of these variables zero. Since the number of steps in each iteration is finite, the algorithm ends in finite time. ∎

It can be checked that the overall complexity of Algorithm 1 is $O(\kappa^2 m^2)$.

B. Multiple trees for the same function

A function may have many possible computation trees. For example, the well-investigated [8] 'sum' function $\Theta(X_1, X_2, X_3) = X_1 + X_2 + X_3$ may be computed by any of the computation sequences $\big((X_1 + X_2) + X_3\big)$, $\big(X_1 + (X_2 + X_3)\big)$, or $\big(X_2 + (X_1 + X_3)\big)$. In general, suppose multiple computation trees $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_\nu$ are given for computing the same function. Let $\mathcal{B}_i$ denote the set of all embeddings of $\mathcal{G}_i$ for $i = 1, 2, \ldots, \nu$. Let $\mathcal{B} = \cup_i \mathcal{B}_i$ denote the set of all embeddings. Under this definition of $\mathcal{B}$, the *Embedding-Edge LP* for this problem is the same as that for a single tree.

One straightforward way to generalize *Node-Arc LP* to multiple trees is to index the edge-sets of the trees by disjoint sets and take flow variables corresponding to all the edges of all trees. Flow conservation equations can be written for each tree, and we need to maximize the sum of the flows generated using such trees. However, such a technique is highly inefficient. For example, $\Theta(X_1, X_2, \ldots, X_\kappa) = X_1 + X_2 + \ldots + X_\kappa$ has $\kappa!$ number of trees, and so the number of variables and constraints will be proportional to $\kappa!$. However, some edges of different trees may represent an identical function of the sources. For example, for the 'sum' function $X_1 + X_2 + X_3 + X_4$, an edge corresponding to the function $X_1 + X_2$ is present in each of the trees corresponding to $\Big(\big((X_1 + X_2) + X_3\big) + X_4\Big)$, $\big((X_1 + X_2) + (X_3 + X_4)\big)$, and $\Big(\big((X_1 + X_2) + X_4\big) + X_3\Big)$. All such edges can be identified and considered as a single flow type. Node-Arc LP can be thus made more efficient by constructing flow constraints for each sub-function rather than each edge of the computation trees. This gives $O(2^\kappa)$ number flow variables instead of $\kappa!$ for the sum function.

The particular function $\Theta(X_1, X_2, \ldots, X_\kappa) = X_1 + X_2 + \ldots + X_\kappa$ is of special theoretical as well as practical interest. Node-Arc LP gives optimal solution with time complexity exponential in κ and polynomial in m. This is not unexpected since the problem is equivalent to the much investigated multicast problem. This is well-known to be NP-hard in κ. Note, however, that our technique suggests a suboptimal technique of considering only a subset of all possible computation trees, which would result in sub-optimal but acceptable performance. The study of tradeoff of restricting embeddings and reducing the overall complexity with suboptimality of the solution, though beyond the scope of this paper, is an interesting avenue for further study.

IV. Discussion and conclusion

In this paper, we have laid the foundations for network flow techniques for distributed function computation. Though we have obtained results for computation trees, we believe that much of our techniques can be extended to larger classes of functions, e.g., fast Fourier transform (FFT), that can be represented by more general graphical structures like directed acyclic graphs and hypergraphs where each edge or hyper-edge represents a distinct function of the sources.

Our computation framework does not allow block coding, i.e., coding across different realizations of the data. Such coding has been used in information theory and network coding literature. We know that, in general, block coding can offer a better computation rates. However, finding the optimal rate and designing optimal coding schemes is a difficult problem under this framework. Further, for undirected multicast networks, it is known that the inter-realization coding can achieve a rate strictly less than twice the rate achieved by routing [11]. It would be interesting to investigate if similar result also holds for function computation over undirected networks.

Altogether, we believe that results in this paper opens many new avenues for further research.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall Inc, 1993.

[2] F. Shahrokhi and D. Matula, "The maximum concurrent flow problem," *J. ACM,*, vol. 37, pp. 318334, 1990.

[3] R. G. Gallager, "Finding parity in simple broadcast networks," *IEEE Trans. on Inform. Th.*, vol. 34, pp. 176–180, 1988.

[4] A. Giridhar and P. R. Kumar, "Computing and communicating functions over sensor networks," *IEEE Jour. on Sel. Areas in Commun.*, vol. 23, no. 4, pp. 755–764, April 2005.

[5] S. Kamath and D. Manjunath, "On distributed function computation in structure-free random networks," in *Proc. of IEEE ISIT*, Toronto, Canada, July 2008.

[6] T. S. Han and K. Kobayashi, "A dichotomy of functions $f(x, y)$ of correlated sources (x, y)," *IEEE Trans. Inform. Th.*, vol. 33, pp. 69–86, 1987.

[7] A. Orlitsky and J. R. Roche, "Coding for computing," *IEEE Trans. Inform. Th.,* vol. 47, no. 3, pp. 903–917, 2001.

[8] B. K. Rai and B. K. Dey, "Sum-networks: system of polynomial equations, reversibility, insufficiency of linear network coding, unachievability of coding capacity," *Submitted to IEEE Trans. Inform. Th., available at http://arxiv.org/abs/0906.0695*.

[9] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing part i : Cut-set bounds," *Submitted to IEEE Trans. Inform. Th., available at http://arxiv.org/abs/0912.2820*.

[10] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *ACM Trans. Algorithms,* vol. 4, pp. 1–17, 2008.

[11] Z. Li and B. Li, "Network coding in undirected networks," in *Proc. of 38th CISS, Princeton, NJ,* Mar. 2004, pp. 257–262.

[12] V. Shah, B. K. Dey and D. Manjunath, "Network Flows for Functions," available at http://arxiv.org/abs/1009.6057 .