Network Flows for Function Computation

Virag Shah, Bikash Kumar Dey, *Member, IEEE*, and D. Manjunath *Member, IEEE*

***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., like 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 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. We then develop a fast combinatorial primal-dual algorithm to obtain near-optimal solutions to these linear programs. As a subroutine for this, we develop an algorithm for finding the minimum cost embedding of a tree in a network with any given set of link costs. We then briefly describe extensions of our techniques to the cases of multiple terminals wanting different functions, multiple computation schemas for a function, computation with a given desired precision, and to networks with energy constraints at nodes.**

***Index Terms*—Function computation; generalized flow conservation; fractional packing; graph embedding.**

I. Introduction

MOTIVATED by sensor network applications, there is significant interest in computing functions of distributed data inside a network. A typical scenario that is considered is the following. Sensor nodes that can make measurements of their environment, perform reasonable amounts of computation, and communicate with other nodes, are distributed in a sensor field. The interest of the sensor network is not so much in the measurement values obtained by the sensors but in some function of these values, say Θ. Some rather simple examples of Θ are the sum, maximum, minimum (or more generally a percentile), sample mean (or more generally sample moments). A more sophisticated example is the Fourier transform of the data. Since the nodes in the network can perform computation, they could participate in the computation of Θ. Thus the interest is in *in-network function computation,* or more expansively, in *distributed computation of a function of distributed data*. We want to study the limit on the computation rate imposed by the network parameters and not by the source data rates; hence we assume that the entire infinite data sequences are readily available at the respective sources at any time. 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 problems (see e.g., [1]–[4]), our work develops such techniques for the first time for function computation.

Early work on in-network computation has been on the asymptotic analysis of the number of transmissions needed to compute specific functions in noisy broadcast networks, e.g., [5]–[7]. Recently, a significant body of work has emerged in which the node locations are assumed to be from a realization of a suitable random point process and nodes can communicate over a shorter distance and not over the entire field. The resulting communication graph of the network is thus a random graph. In this setting a probabilistic characterization of the asymptotic (in the number of nodes) computation rate for different classes of functions, e.g., 'type-threshold functions' and 'type sensitive functions' [8], have been obtained, e.g., [8]–[11].

Another class of work considers simpler networks with a small number of correlated sources [12]–[16]. Much of this work takes the information theoretic perspective in which the objective is to find encoding rate regions for reliably communicating the desired function. Yet another approach to function computing is the recent work in network coding literature [17]–[19]. Here larger and more complex networks with independent sources are considered. However, designing optimal coding schemes and finding capacity is a difficult problem except for very special functions or networks, e.g., [17], [18].

In this paper we make a significant departure from the above. We consider arbitrary computable functions of the distributed data for which a computation schema is described by a directed tree. A computation schema defines a sequence of operations to compute the function. An arbitrary capacitated communication network—each link having a capacity—is also assumed given. The function Θ for each element of the sequence of data is to be computed over the network using the schema at the highest rate possible in the network. In this paper we develop and analyze algorithms that determines the communication and computation sequence in the network to compute Θ at the highest possible rate. In our setup, we assume unbounded memory and computational power at each node. Our algorithms and protocols require a memory-size and computational power that depend on the sizes of the network and the computation tree. We also do not assume any latency requirement on the computation. Even so, the

Manuscript received April 10, 20012, accepted December 19, 2012. This work was supported in part by Bharti Centre for Communication at IIT Bombay and a project from the Department of Science and Technology (DST), India. The material in this paper was presented in parts at IEEE International Symposium on Information Theory 2011, Saint-Petersburg, Russia, and IEEE Global Communications Conference 2011, Houston, Texas, USA.

V. Shah was with the Department of Electrical Engineering, Indian Institute of Technology Bombay, India during the course of this work. He is now with the University of Texas, Austin (e-mail: virag4u@gmail.com).

B. K. Dey and D. Manjunath are with the Department of Electrical Engineering, Indian Institute of Technology Bombay, India (e-mail: {bikash,dmanju}@ee.iitb.ac.in).

Digital Object Identifier 10.1109/JSAC.2013.130409.

computational delay at nodes are not directly relevant under the preceding assumptions because handling such delays only requires additional buffer at the nodes to sychronize incoming data.

Our techniques are applicable to networks with directed links, networks with undirected links, and networks with both directed and undirected links with capacity constraints; however, we present our results only for networks with undirected links. We also restrict to wireline networks for explaining our work even though some of our techniques can be adapted to work in standard wireless network models. Our work has significant relation to the well-studied problem of fractional Steiner-tree packing which provides a solution to the routing multicast problem—finding the best multicast rate under routing [20]. The exact relation is explained in Extension 1 in Section V. There are some similarities of our work with that of graph embedding, e.g., [1], [21], [22] but there are significant differences in the modeling assumptions and in the embedding objectives. Such work on embedding typically assume the target network to be a 'regular network' like a hypercube or a mesh and all link capacities are assumed equal. In a subsequent work [23], it is shown that computation in a network without block coding, as is the focus of this paper, is near-optimal for a wide class of function computation problems.

A. An Example and Motivation

Let us consider the function $\Theta(X_1, X_2, X_3) = X_1 \cdot X_2 + X_3$ of three variables X_1, X_2, and X_3, generated at three sources s_1, s_2, and s_3 respectively. A terminal node t is required to obtain the function $\Theta(X_1, X_2, X_3)$. We assume that all the three data symbols are from the same alphabet $\mathcal{A}$, and multiplication $(\cdot)$ and addition $(+)$ operations result in a symbol in the same alphabet. The computation of the function can be broken into two parts—first compute $(X_1 \cdot X_2)$ and then add X_3 to the product. These two operations can be done at different nodes in the network in the above order. This decomposition of the computation can be represented by the graph shown in Fig. 1(a). Such a graphical representation of the computation will henceforth be called a computation tree. Note that such a decomposition into sub-computations is used while computing even in a single sequential computer. Clearly, each edge of the computation tree represents a *unique* function of the source symbols corresponding to its ancestor source nodes, e.g., edge corresponding to $X_1 \cdot X_2$ represents a function of the symbols generated at the ancestor sources s_1 and s_2.

Now consider computing $\Theta(X_1, X_2, X_3)$ in the network shown in Fig. 1(b) where each edge has unit capacity. There are multiple ways of receiving this function at the terminal t and they depend on what computations are performed at which nodes and also the paths chosen for the data flows. Two such ways of computing this function are shown in Figs. 1(c) and 1(d). Each of these graphs show the nodes at which each sub-computation would be performed and also the path along which each source-data or computed data would flow. Equivalently, it associates (1) a network node with each node in the computation tree, and (2) a directed path in the network with each edge in the computation tree. We call such a directed graph representing a possible way of computing the function according to the computation tree in the given network as an *e*mbedding. An embedding of the computation tree in a network is formally defined in Section II. Clearly, an intelligent time-sharing between the various embeddings may give a higher number of computations of Θ per use of the network, on average, than when only one such embedding is used. This raises the natural question: how to allocate the timeshare for every possible embedding to achieve the maximum rate at which Θ can computed in a given network? In this paper, we address this question.

Time-sharing is a standard technique in information theory, and is implicit in the standard packing formumation in multi-commodity flow problems. In multi-commodity problems, different paths are time-shared for a given flow to maximize the total flow under the capacity constraints of the links. In our computation problem, since a computation is performed over an embedding, it is natural to time-share between such embeddings. However, in contrast to multi-commodity flow problems, in our setup different data-flows from a particular realization need to be synchronized so that the computations are restricted to data of the same realization. This synchronization requires careful design of a scheduling protocol. One way of doing this is explained in Appendix B.

B. Organization and Summary of Contributions

We begin by describing the model and definitions in detail in the next section. Sections III and IV present the main contributions of this paper. In Section III, we first present a 'natural' linear program (LP), *Embedding-Edge-LP,* that, as stated in Theorem 1, optimally allocates flows for each of the embeddings and achieves the maximum rate possible under our setup. We will see that this LP has exponential complexity in terms of the network parameters. We then present a second LP, called *Node-Arc-LP,*, which is based on a suitably defined flow conservation principle. This LP outputs a set of 'consistent' link flows and it can be solved in polynomial time and space. However, its solution does not directly tell us the timeshare for the embeddings, and thus does not directly give a routing-computation scheme. Such a timeshare of the embeddings is obtained by an algorithm *Extract-Embeddings* that converts the flow rates obtained from *Node-Arc-LP* into a flow allocation on the embeddings. Thus *Node-Arc-LP* and *Extract-Embeddings* together provides a solution to *Embedding-Edge-LP* . We show that this algorithm also has polynomial time and space complexity. In Section IV, we present *FANO* (FAst Near-Optimal algorithm), a fast primal-dual algorithm which, for a given $\epsilon > 0$, finds a solution to achieve at least $(1 - \epsilon)$ fraction of the optimal rate of computation. We call such a solution an $(1 - \epsilon)$-optimal solution. This algorithm uses an oracle subroutine which finds a minimum cost embedding of the computation tree in the network. We provide an efficient algorithm, *Min-Cost-Embedding*, to obtain the same. This algorithm is also of independent interest. In Section IV-A, we discuss the distributed implementation of our near-optimal algorithm. Appendix A gives a proof of Theorem 1. Once we know an optimal timeshare of different embeddings, we need

to convert the allocation into a usable schedule of computation at the nodes and communication along the edges. A procedure to derive this schedule from the timeshare allocations is given in Appendix B.

Four interesting extensions of our results are presented in Section V. First, we allow multiple computing schema for computing the same function. Then we consider multiple terminals computing distinct functions of disjoint sets of sources. For this problem, we show how we can modify our techniques to maximize either the weighted sum-rate of computations, or maximize the rate-vector in a given direction to compute the rate-vectors at the boundary of the rate-region. In the third extension, we consider the problem of computing a function with a desired precision which is achieved by allowing possibly different precision for each type of data. In the fourth extension, we consider a network with energy-constrained nodes, and assume that each type of data, i.e., each edge of the computation tree, requires some fixed but different amount of energy to compute/generate, transmit, and receive.

II. The model, notation, and definitions

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 $e \in E$ represents a half duplex link with a total non-negative capacity $c(e)$. Whenever we need to refer to the incident nodes, say u and v, the edge between u and v will be called (u, v) or uv. This notation will also be specially used to denote an edge when communication in a specific direction (from u to v) is to be indicated.

In the network $\mathcal{N}$, $S = \{s_1, s_2, \ldots, s_\kappa\} \subset V$ is the set of κ source nodes, and t is the terminal node. 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 $\mathcal{A}$. The link capacities are expressed in an $|\mathcal{A}|$-ary unit. X_i is used to denote a representative element of the sequence. Let $\mathbf{X} \triangleq [X_1, \ldots X_\kappa]$, and its n-th realization $\mathbf{X}(n) \triangleq [X_1(n), \ldots, X_\kappa(n)]$. Without loss of generality, we assume that each source node in the network generates exactly one data sequence; if a source node generates two or more data sequences then this can be represented by multiple source nodes connected by infinite capacity links. We also assume that there is only one terminal node. For any positive integer l, we denote $\{1, 2, \cdots, l\}$ by $[1, l]$.

A given function $\Theta : \mathcal{A}^\kappa \to \mathcal{A}$ of $\mathbf{X}$ needs to be obtained at the terminal node t for each k at the highest possible rate. A computation schema for Θ is given and is 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 labelled $\mu_1, \mu_2, \ldots, \mu_{|\Omega|}$ 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 $\mathbf{X}$. 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 node μ_i to node 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. (An out-degree of greater than one will result in a directed acyclic graph (DAG), which is outside the scope of this work.) The elements of Γ are labelled $\theta_1, \theta_2, \ldots, \theta_{|\Gamma|}$ with $\theta_1, \theta_2, \ldots, \theta_\kappa$ being the outgoing edges from $\mu_1, \mu_2, \ldots, \mu_\kappa$ respectively, and $\theta_{|\Gamma|} = \Theta$ 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 edge θ_j to the tail node of edge θ_i. The nodes and edges of $\mathcal{G}$ can be labeled as above in $O(|\Gamma|) = O(\kappa)$ time.

For any edge $\theta \in \Gamma$, let $\mathsf{tail}(\theta)$ and $\mathsf{head}(\theta)$ represent, respectively, the tail and the head nodes of edge θ. Let $\Phi_\uparrow(\theta)$ and $\Phi_\downarrow(\theta)$ denote, respectively, the immediate predecessors and the immediate successors of θ, i.e.,

$$\begin{aligned}\Phi_\uparrow(\theta) &\triangleq \{\eta \in \Gamma | \mathsf{head}(\eta) = \mathsf{tail}(\theta)\} \text{ and}\\ \Phi_\downarrow(\theta) &\triangleq \{\eta \in \Gamma | \mathsf{tail}(\eta) = \mathsf{head}(\theta)\}.\end{aligned}$$

Each edge θ of $\mathcal{G}$ represents a unique function of $\mathbf{X}$ that can be computed from the functions corresponding to the edges in $\Phi_\uparrow(\theta)$. Further, each function takes values from the same alphabet $\mathcal{A}$. Throughout this paper, except in the extension on DAG computation schema in Sec. V, $\Phi_\downarrow(\theta)$ contains exactly one edge if $\theta \neq \theta_{|\Gamma|}$. For $\theta = \theta_{|\Gamma|}$, it is always true that $\Phi_\downarrow(\theta_{|\Gamma|}) = \emptyset$.

Remark *1:* Each intermediate function of $\mathbf{X}$ taking values from the same alphabet is not unreasonable even when all the computations are over real numbers because computations are performed using a fixed precision. However, it may be reasonable in some applications to use different precision for different intermediate functions (edges of $\mathcal{G}$). This can be handled in a simple manner in our setup, and is explained in an extension presented in Section V.

All computations at any node of the network are restricted to conform to the given computation tree in the following sense. Any node in the network is allowed to compute only the functions represented by the edges of $\mathcal{G}$. A node computing a function θ of any realization of the data symbols $\mathbf{X}(n)$ must do so by using the functions $\Phi_\uparrow(\theta)$ of $\mathbf{X}(n)$.

Let $\aleph(v) \triangleq \{u \in V | uv \in E\}$ denote the set of neighbors of a node $v \in V$ and let $\aleph'(v) \triangleq \aleph(v) \cup \{v\}$ be the set of neighbors of v including itself. A sequence of nodes $v_1, v_2, \cdots, v_l$, $l \geq 1$, is called a path if $v_i v_{i+1} \in E$ for $i = 1, 2, \ldots, l-1$. and $v_i \neq v_j$ for $1 \leq i < j \leq l$. Note that according to our definition, a single node also qualifies as a path. Also, our paths are directed. The set of all paths in $\mathcal{N}$ is denoted by $\mathcal{P}$. With usual abuse of notation, for such a path P, we will say $v_i \in P$ and also $v_i v_{i+1} \in P$. The nodes v_1 and v_l are called, respectively, the start node and the end node of P, and are denoted as $\mathsf{start}(P)$ and $\mathsf{end}(P)$.

As discussed in Section 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 *1:* An embedding is a mapping $B : \Gamma \to \mathcal{P}$ such that

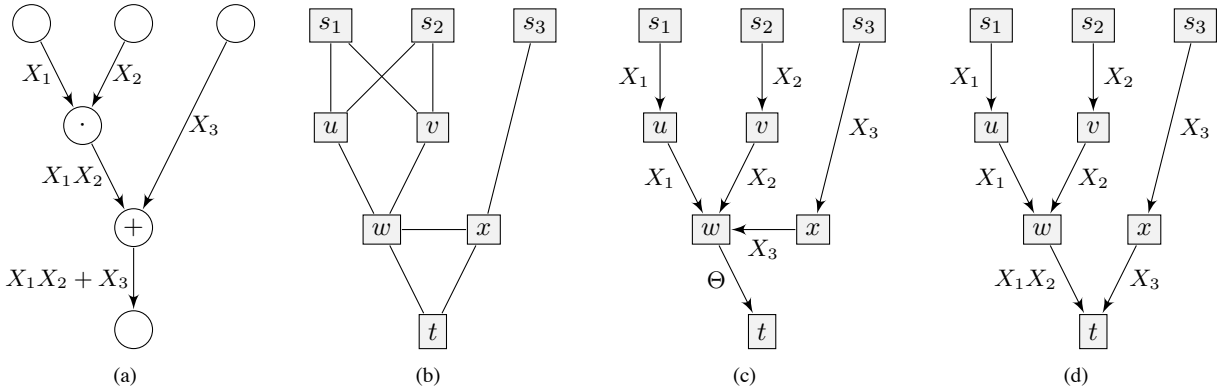


Fig. 1. Computing $\Theta = X_1X_2 + X_3$ over a network. (a) A schema to compute Θ. (b) A network to compute $\Theta = X_1X_2 + X_3$. (c) A possible embedding that computes Θ. (d) An alternative embedding.

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

Note that, an embedding B maps every edge θ_l of the computation tree to a *directed* path $B(\theta_l)$ in the network. The image is a directed path even if the network is undirected. The path $B(\theta_l)$ in $\mathcal{N}$ carries the data θ_l, the node $\mathsf{start}(B(\theta_l))$ generates or computes the data θ_l, and the node $\mathsf{end}(B(\theta_l))$ uses θ_l to compute some other function or, if $l = |\Gamma|$ then, simply receives it. In the special case when $B(\theta_l)$ is a single node (which, by our definition, is a path as well), that node generates/computes θ_l but does not forward it to other links. It uses θ_l to compute some other function (specifically $\Phi_\downarrow(\theta_l)$) or, if $l = |\Gamma|$ then, simply receives it.

Example *1:* (i) For the embedding shown in Fig. 1(c), $B(X_1) = s_1uw$, $B(X_2) = s_2vw$, $B(X_3) = s_3xw$, $B(X_1X_2) = w$, and $B(X_1X_2 + X_3) = wt$.

(ii) For the embedding shown in Fig. 1(d), $B(X_1) = s_1uw$, $B(X_2) = s_2vw$, $B(X_3) = s_3xt$, $B(X_1X_2) = wt$, and $B(X_1X_2 + X_3) = t$.

It is worth noting that, a computation tree, and thus its embeddings, ignores the operations represented by its nodes. Thus the same computation tree may represent computation of different functions by changing the operations represented by its nodes. For example, Fig. 1(a) is also a computation tree for the function $\Theta = (X_1 + X_2)X_3$, or $\Theta = (X_1 + X_2)/X_3$, or many such other functions. In fact, all our subsequent developments depend only on the computation tree, and not on the particular function.

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 $r_B(e) = |\{\theta \in \Gamma | e \text{ is a part of } B(\theta)\}|$.

Example *2:* Fig. 2(a) shows a computation tree, which represents the natural sequence of operations involved in the computation of functions like $\Theta = (X_1X_2 + X_3)X_4$. Fig. 2(b) shows an undirected network where s_1t has capacity 2 and all the other edges have capacity 1. In this network, clearly an efficient embedding, which achieves a computation rate of 1, is (as shown in Fig. 2(c)) $B(X_1) = s_1ts_2$, $B(X_2) = s_2$, $B(X_3) = s_3$, $B(X_4) = s_4$, $B(X_1X_2) = s_2s_3$, $B(X_1X_2 + X_3) = s_3s_4$, and $B(\Theta) = s_4s_1t$. In this embedding, the edge s_1t is used twice–once in $B(X_1)$ and once in $B(\Theta)$.

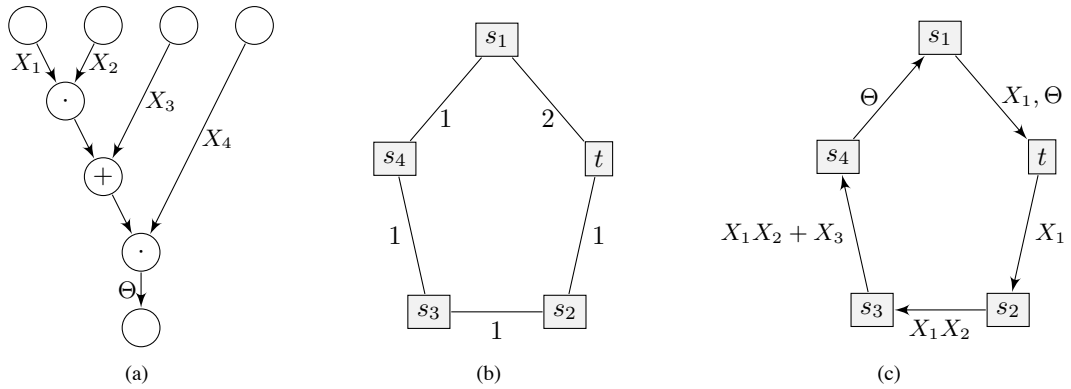
An edge may also be used to carry flows on different embeddings. Therefore in an assignment of flows on different embeddings, i.e., in a particular timesharing scheme, the edge may carry multiple types of data (i.e., different functions of $\mathbf{X}$) of different amounts. Also note that, if $\mathsf{start}(B(\theta_i)) = \mathsf{end}(B(\theta_i))$, 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.

Recall that we restrict the computations in the network to be as dictated by the given computation tree. In an embedding, some particular network nodes play the roles of the nodes of the computation tree, and the function θ flows along the path $B(\theta)$. Since different realizations of data are never 'mixed' in the network, if one traces one realization of data (and their functions) in the network, it must clearly flow on an embedding of the computation tree. However, different embeddings may be used for different realizations. This flexibility allows the time-sharing of the embeddings to achieve a better computation rate under capacity constraints of the edges. In the following, we describe our transmission and computation model in more detail and precision.

A. The communication and computation model

We now give the basic assumptions in our communication and computation model. The most important assumptions are 5 and 6, which state that no mixing of data is allowed across realizations of data, and that any computation at any node should be according to the operations dictated by the computation graph.

1) Time is slotted, and all links can be used simultaneously once in each slot.
2) There is unbounded memory at each node.
3) Source s_i generates the data sequence $\{X_i(k)\}_{k\geq 1}$. The entire data sequence is assumed to be available at any time at s_i. This is to ensure that the computation rate is restricted only by the link capacities, and not by the source data rates.
4) If the capacity of a link e is $c(e)$ per use (that is, per slot), and if n is any positive integer, then in any consecutive block of n time-slots, a total of upto $\lfloor nc(e) \rfloor$ symbols can be communicated across e.



(a) (b) (c)

Fig. 2. A computation tree of, for example, $\Theta = (X_1X_2 + X_3)X_4$. (a) A network to compute Θ, the numbers labeling the edges indicate their capacity (b) A possible embedding that computes Θ at unit rate.

5) Suppose $\mathcal{Y} = \cup_{n\in\mathbb{N}}\mathcal{Y}_n$ is the data available at a node u in the network in the beginning of a time-slot, where $\mathcal{Y}_n \subseteq \{\theta(\mathbf{X}(n))|\theta \in \Gamma\}$ denotes the set of functions of the sources' data of realization n available at the node. Then any symbol communicated in that time-slot on an outgoing link is a function $g(\mathcal{Y})$ of the available data[1]. Further, for some $n \in \mathbb{N}$, the function $g(\mathcal{Y})$ is of the form
$$g(\mathcal{Y}) = g(\mathcal{Y}_n),$$
that is, it is a function of only one realization of data generated at the sources.
6) One of the following holds.
A) $g(\mathcal{Y}_n) \in \mathcal{Y}_n$, i.e., the symbol transmitted on the link is one of the symbols received at that node.
B) $g(\mathcal{Y}_n) = \theta(\mathbf{X}(n))$ for some $\theta \in \Gamma$ such that $\{\eta(\mathbf{X}(n))|\eta \in \Phi_\uparrow(\theta)\} \subseteq \mathcal{Y}_n$. That is, the transmitted symbol is computed from the data available at the node using an operation indicated by a node in Ω.

The conditions 5) and 6) constrains the computations at nodes to be restricted among the same realization of data, and also such computations to follow the operations indicated by the computation graph.

B. Definitions

In this subsection, we formally define a routing-computing scheme, achievable rate, and computing capacity. These are natural definitions under our set-up, and a reader preferring to skip the formalism can skip these definitions and the proof of Theorem 1 in Appendix A.

Let us consider a fixed block of K source symbols indexed by $1, 2, \cdots, K$ for each source. At any point of time, a node can have in its memory, a subset of the universe of data: $\mathcal{D} = \Gamma \times [1, K]$, where any element $(\theta, k) \in \mathcal{D}$ denotes the function $\theta(\mathbf{X}(k))$ of the k-th data symbols. For any subset $\mathcal{D}_1 \subseteq \mathcal{D}$ and any $k \in [1, K]$, we define
$$\mathcal{D}_1(k) = \{\theta \in \Gamma|(\theta, k) \in \mathcal{D}_1\}$$
as the cross-section of $\mathcal{D}_1$ at k.

Definition *2:* A subset $\Gamma' \subseteq \Gamma$ is said to be *irreducible* if $\nexists\theta \in \Gamma$ such that $\Phi_\uparrow(\theta) \subseteq \Gamma'$. A subset $\mathcal{D}_1 \subseteq \mathcal{D}$ is said to be irreducible if for each $k \in [1, K]$, $\mathcal{D}_1(k)$ is irreducible.

We now define a routing-computing scheme below. A scheme can be thought in terms of L events $E_l, 1 \le l \le L$; and $\mathcal{D}_{v,l} \subseteq \mathcal{D}$ and $\mathcal{D}_{v,l+1} \subseteq \mathcal{D}$ as the state (of knowledge) of a node v before and after the l-th event E_l respectively. For each $l \le L$, the event E_l is either a computation event where a node v computes a function $\theta(\mathbf{X}(k))$ using $\{\eta(\mathbf{X}(k))|\eta \in \Phi_\uparrow(\theta)\}$, which is available at v, or a communication event where some data $\theta(\mathbf{X}(k))$ is communicated from a node u to another node v over the edge uv. In the case of a computation event at v, the states of all other nodes are unchanged by E_l. In the case of a communication event from u to v, the states of all nodes other than u and v are unchanged by E_l.

Definition *3:* A $(\{N_e|e \in E\}, K)$ routing-computing scheme for $(\mathcal{N}, \mathcal{G})$ has for some positive integer L, some irreducible[2] subsets $\{\mathcal{D}_{v,l} \subseteq \mathcal{D}|v \in V, 1 \le l \le L+1\}$ ($\mathcal{D}_{v,l}$ denotes the subset of $\mathcal{D}$ available at node v at time l) such that

1. *Initial condition:* For $1 \le i \le \kappa$, $\mathcal{D}_{s_i,1} = \{(\theta_i, k)|1 \le k \le K\}$. For all $v \in V \setminus \{s_i|1 \le i \le \kappa\}$, $\mathcal{D}_{v,1} = \emptyset$.

2. For each $l < L + 1$, one of the following holds.

(i) *Computation event:* For some $v \in V, k \in [1, K]$ and $\theta \in \mathcal{D}_{v,l}(k)$ such that $\Phi_\uparrow(\theta) \subseteq \mathcal{D}_{v,l}(k)$, it holds that
$$\mathcal{D}_{v,l+1} = \{(\theta, k)\} \cup \mathcal{D}_{v,l} \setminus \{(\eta, k)|\eta \in \Phi_\uparrow(\theta)\},$$
$$\mathcal{D}_{u,l+1} = \mathcal{D}_{u,l}, \quad \forall u \in V \setminus \{v\}.$$
Here, v computes the function $\theta(\mathbf{X}(k))$ using $\{\eta(\mathbf{X}(k))|\eta \in \Phi_\uparrow(\theta)\}$, which is available at v.

(ii) *Communication event:* For some $uv \in E, \theta \in \Gamma$ and

[1]More precisely, the available data should be indexed by the time-slot and u, and the function should be indexed by the time-slot, the link, and the symbol-index in that time-slot in case of multiple symbols transmitted in a time-slot. However, we omit these indices to keep the notation simple.

[2]Irreducibility of these subsets is not a crucial requirement. However, under the routing-computing setup we consider, no node needs to retain in its memory the values it has used to compute or it has transmitted. So, it is sufficient to assume that the nodes always has, in its memory, an irreducible subset of $\mathcal{D}$.

$k \in [1, K]$ such that $(\theta, k) \in \mathcal{D}_{u,l}$, it holds that

$$\mathcal{D}_{u,l+1} = \mathcal{D}_{u,l} \setminus \{(\theta, k)\}$$
$$\mathcal{D}_{v,l+1} = \mathcal{D}_{v,l} \cup \{(\theta, k)\}$$
$$\mathcal{D}_{w,l+1} = \mathcal{D}_{w,l} \quad \forall w \neq u, v.$$

Here, $\theta(\mathbf{X}(k))$ is communicated from u to v on the link uv.

3. *Final condition:*

$$\mathcal{D}_{t,L+1} = \{(\Theta, k) | 1 \leq k \leq K\},$$
$$\mathcal{D}_{v,L+1} = \emptyset \quad \forall v \neq t.$$

4. *Total link usage:* For all $e \in E$,

$$|\{l \in [1, L] | E_l \text{ is a communication over } e\}| = N_e.$$

We emphasize again that our schemes only combine symbols with same k (the same realization) for the purpose of computation as dictated by the computation graph. So we call our schemes *routing-computing schemes*. They do not allow combining different realizations of data at a node—a flexibility which may offer higher computation rates. See Sec. VI for an example.

A $(\{N_e | e \in E\}, K)$ routing-computing scheme makes $N_e/c(e)$ uses of an edge $e \in E$ (that is, the edge needs to be used for $N_e/c(e)$ slots), and achieves K computations of Θ at the terminal. Note that if a computation rate λ is achieved by a scheme, then no edge is used for more than K/λ slots. This motivates the following definition.

Definition *4:* For a given capacity constrained network $\mathcal{N}$, $\{c(e) | e \in E\}$, and a computation tree $\mathcal{G}$, A rate λ is said to be $(\mathcal{N}, \mathcal{G})$-*achievable* if for every $\epsilon > 0$, there is a $(\{N_e | e \in E\}, K)$ routing-computing scheme for $(\mathcal{N}, \mathcal{G})$ so that $N_e(\lambda - \epsilon) \leq c(e)K, \ \forall e \in E$.

Definition *5:* The supremum of all $(\mathcal{N}, \mathcal{G})$-achievable rates is called the *computing capacity* for $(\mathcal{N}, \mathcal{G})$, and is denoted by $C(\mathcal{N}, \mathcal{G})$.

Definition *6:* An algorithm is said to give a near-optimal solution if, for any chosen $\epsilon > 0$, it produces a scheme which achieves at least $(1 - \epsilon)$ fraction of the computing capacity.

III. Efficient flow-based algorithm

In this section, we present the first part of our main contributions.

- In Section III-A, we give a basic linear program, the *Embedding-Edge-LP* , which characterizes the computing capacity $C(\mathcal{N}, \mathcal{G})$ of a network $\mathcal{N}$ for computing a function using a given computation tree $\mathcal{G}$.
- In Section III-B, we give an alternate LP, the *Node-Arc-LP* , that can be solved in polynomial time. We then present an algorithm which obtains a solution of the *Embedding-Edge-LP* with the same rate from a solution of the *Node-Arc-LP* .

A. The Embedding-Edge LP

As discussed in Sections I and II, the function for a particular sample of the data can be computed over the network using any embedding of $\mathcal{G}$ in $\mathcal{N}$. Let $\mathcal{B}$ be the set of all embeddings of $\mathcal{G}$ in $\mathcal{N}$. 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. Thus, for a given allocation of the timeshares to each of the embeddings, λ, defined by

$$\lambda \triangleq \sum_{B \in \mathcal{B}} x(B)$$

is the rate at which Θ can be computed in the network. This leads us to formulate *Embedding-Edge LP*, a linear program to maximize the computation rate of Θ. 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 the following constraints.

1. Capacity constraints

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

2. Non-negativity constraints

$$x(B) \geq 0, \ \forall \ B.$$

The following theorem states that *Embedding-Edge-LP* characterizes the computing capacity $C(\mathcal{N}, \mathcal{G})$.

Theorem *1:* The maximum value of λ obtained by a solution of the *Embedding-Edge-LP* is the computing capacity $C(\mathcal{N}, \mathcal{G})$.

We remark that the *Embedding-Edge LP* is essentially a fractional packing problem—packing of the embeddings of $\mathcal{G}$ into $\mathcal{N}$. Similar packing LPs characterize the maximum achievable rates in multi-commodity problems. However, since our problem is more general (computing problem in contrast to communication problem), we explicitly present a proof of Theorem 1 in Appendix A in two parts: (i) $(\mathcal{N}, \mathcal{G})$-achievability of the rate obtained by solving the LP, and (ii) a proof of converse, i.e., that no higher rate is $(\mathcal{N}, \mathcal{G})$-achievable.

For a given allocation $\{x(B)\}$, on each edge of $\mathcal{N}$, the flows (different functions of $\mathbf{X}$) have to be mixed according to different embeddings but we need to be careful to not mix different realizations of the data stream. Thus, in an implementation, for a given $\{x(B)\}$, we need to carefully devise a protocol to schedule the computation at the nodes and the communication on the edges in such a way that data from different realizations (i.e., different elements of the vector sequence $\{\mathbf{X}(k)\}_{k \geq 0}$) are not mixed. Such a schedule is presented in Appendix B.

The cardinality of $\mathcal{B}$ is, in general, exponential in $|V|$. Hence a naive solution to *Embedding-Edge LP* will have an exponential complexity in the network parameters and an efficient solution will need to exploit some structure of the problem. In the following subsection, we present an LP based on a flow-conservation principle which can be solved in polynomial time.

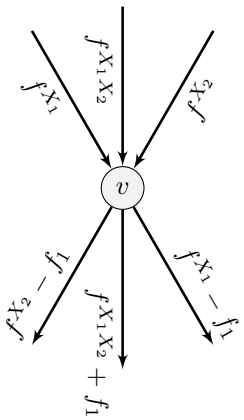


Fig. 3. Functional flow conservation at a node with three input and three output flows.

B. The Node-Arc LP

In multi-commodity flow problems, a simple flow-conservation principle applied at each node gives a polynomial-time solution to the problem which is otherwise hard to solve from the Path-Edge LP (similar to our *Embedding-Edge-LP*). Traditional flow-conservation does not hold in our problem as explained subsequently, and this poses a difficulty of ready use of the same technique. In this subsection, we point out a flow conservation that holds in our setup and present an LP for our problem based on this. This will finally allow a polynomial time solution of the problem.

In our setup, flows can be consumed as well as generated at the nodes. This is illustrated in Fig. 3. In this figure, Node v has f^{X_1}, f^{X_2}, and $f^{X_1X_2}$ units of incoming flow of X_1, X_2, and X_1X_2 respectively. Suppose it uses f_1 units of X_1 and X_2 each to compute f_1 units of X_1X_2. Thus the corresponding outgoing flow from v consists of $(f^{X_1} - f_1)$ units of X_1, $(f^{X_2} - f_1)$ units of X_2, and $(f^{X_1X_2} + f_1)$ units of X_1X_2. This violates the traditional flow-conservation principle, but we develop a more general conservation principle which allows such flow values resulting from computation at nodes.

We first assume that each node in the network has a 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 a source or the values that are computed at a node. For example, for computing $\Theta = X_1X_2 + X_3$ according to the computation tree in Fig. 1(a), if a node computes $X_1 \cdot X_2$ from the X_1 and X_2 it receives, and then computes $X_1 \cdot X_2 + X_3$ by using $X_1 \cdot X_2$ that it computed and X_3 that it receives, then both $X_1 \cdot X_2$ and $X_1 \cdot X_2 + X_3$ will be assumed to flow in the self-loop at the node.

The variables in the *Node-Arc LP* are

$$\left\{f^{\theta}_{uv}, f^{\theta}_{vu} | uv \in E, \theta \in \Gamma\right\} \cup \left\{f^{\theta}_{uu} | u \in V, \theta \in \Gamma\right\} \cup \{\lambda\},$$

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.

As we described in the example above, the flow conservation rule accounts for the fact that an intermediate node in $\mathcal{N}$ can (1) forward the flows it receives, and (2) generate a flow of type θ on its self-loop by terminating (consuming) equal amounts 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 $\theta_{|\Gamma|}$. An example of the flows on the edges and the self-loops corresponding to a flow assignment on two embeddings is shown in Fig. 4. The *Node-Arc LP* is as follows. Recall that $\aleph'(v)$ denotes the set of neighbors of v and itself.

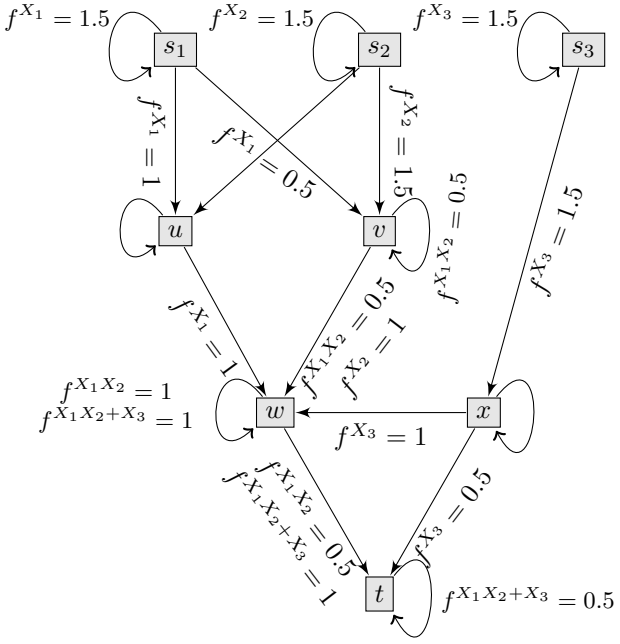


Fig. 4. An example of edge-flow values for a total computation rate of 1.5.

Node-Arc-LP

Maximize λ

subject to the following constraints.

1. Functional conservation of flows at all nodes $v \in V$,

$$f^{\eta}_{vv} + \sum_{u \in \aleph(v)} f^{\theta}_{vu} - \sum_{u \in \aleph'(v)} f^{\theta}_{uv} = 0, \quad \forall \theta \in \Gamma \setminus \{\theta_{|\Gamma|}\} \text{ and } \forall \eta \in \Phi_{\downarrow}(\theta). \tag{1}$$

2. Conservation and termination of $\theta_{|\Gamma|}$ at all nodes $v \in V$,

$$\sum_{u \in \aleph(v)} f^{\theta_{|\Gamma|}}_{vu} - \sum_{u \in \aleph'(v)} f^{\theta_{|\Gamma|}}_{uv} = \begin{cases} -\lambda & \text{for } v = t, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

3. Generation of θ_l $\forall l \in \{1, 2, \ldots, \kappa\}$ and at all nodes $v \in V$,

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

4. Capacity constraints at all edges $uv \in E$,

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

5. Non-negativity constraints,

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

The functional conservation of flows used in the *Node-Arc-LP* is a key concept in this paper. In the following example, we explain how the flows indicated in Fig. 4 respect functional conservation of flows.

Example *3 (Functional conservation of flows):* (i) Consider node w, $\theta = X_1X_2$, and $\eta = X_1X_2 + X_3$. The first term in the LHS of (1) is $f^{X_1X_2+X_3}_{ww} = 1$. The second term is $f^{X_1X_2}_{wu} + f^{X_1X_2}_{wv} + f^{X_1X_2}_{wx} + f^{X_1X_2}_{wt} = 0+0+0+0.5 = 0.5$, and the third term is $f^{X_1X_2}_{ww} + f^{X_1X_2}_{uw} + f^{X_1X_2}_{vw} + f^{X_1X_2}_{xw} + f^{X_1X_2}_{tw} = 1 + 0 + 0.5 + 0 + 0 = 1.5$. Thus clearly, they satisfy (1).

(ii) Consider the same flows $\theta = X_1X_2$ and $\eta = X_1X_2 + X_3$ at the node t. The first term in the LHS of (1) is $f^{X_1X_2+X_3}_{tt} = 0.5$. The second term is $f^{X_1X_2}_{tw} + f^{X_1X_2}_{tx} = 0 + 0 = 0$, and the third term is $f^{X_1X_2}_{tt} + f^{X_1X_2}_{wt} + f^{X_1X_2}_{xt} = 0 + 0.5 + 0 = 0.5$. Clearly, they too satisfy (1).

(iii) Consider the flow $\theta = X_1X_2 + X_3$ at w. The first term in the LHS of (2) is $f^{X_1X_2+X_3}_{wu} + f^{X_1X_2+X_3}_{wv} + f^{X_1X_2+X_3}_{wx} + f^{X_1X_2+X_3}_{wt} = 0 + 0 + 0 + 1 = 1$. The second term is $f^{X_1X_2+X_3}_{ww} + f^{X_1X_2+X_3}_{uw} + f^{X_1X_2+X_3}_{vw} + f^{X_1X_2+X_3}_{xw} + f^{X_1X_2+X_3}_{tw} = 1 + 0 + 0 + 0 + 0 = 1$. Clearly they satisfy (2).

(iv) Consider the flow $\theta = \Theta = X_1X_2 + X_3$ at t. Note that in this example, $\lambda = 1.5$. The first term in the LHS of (2) is $f^{X_1X_2+X_3}_{tw} + f^{X_1X_2+X_3}_{tx} = 0 + 0$. The second term is $f^{X_1X_2+X_3}_{tt} + f^{X_1X_2+X_3}_{wt} + f^{X_1X_2+X_3}_{xt} = 0.5 + 1 + 0 = 1.5$. Clearly, they too satisfy (2).

The *Node-Arc-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.

A solution of the above LP gives a set of flow values on each link. Note that unlike a multi-commodity flow problem, the solution to *Node-Arc-LP* does not readily describe a practical communication and computation protocol over the network. In the multi-commodity 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 over which link. However, in the function computation problem a node cannot do this arbitrarily; only data of the same realization can be mixed at network nodes as per the computation schema. We thus present *Extract-Embeddings* algorithm, which, from any feasible solution of *Node-Arc-LP* , obtains a feasible solution for *Embedding-Edge-LP* that achieves the same total flow λ. As pointed out later, *Extract-Embeddings* constructs only a polynomial number of embeddings and assigns non-zero flows to them. Thus *Extract-Embeddings* does not enumerate all possible embeddings, as would be the case for solving *Embedding-Edge LP* directly.

Before discussing *Extract-Embeddings* we first remark that a cyclic flow, i.e., a flow of non-zero volume of any type θ across a cycle in the network graph, can occur in a feasible solution of the *Node-Arc LP*. This is clear from the formulation, and such extra non-contributing cyclic flows are standard artifacts in the flow-based solutions of multi-commodity problems as well. Such a cyclic flow can be removed from the solution without affecting its feasibility or its objective value λ.

Extract-Embeddings is an iterative algorithm. 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 $f^{\theta_{|\Gamma|}}_{tt} > 0$, 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 $f^{\theta_{|\Gamma|}}_{uu} > 0$, and assign the explored path $u \ldots t$ 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 $z(\cdot)$ keep track of the maximum flow that the embedding can carry; this 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 cyclic flow of type θ_i. If such a cycle is found, it is removed from the explored path and the corresponding flow volume is removed from the edges in the cycle. The flow removed from an embedding or a cycle is the maximum possible flow on that embedding or cycle, so that when removed, the corresponding flow in one link (the bottleneck link) is made zero by the removal.

Theorem *2:* Algorithm *Extract-Embeddings* is correct and the time complexity is in $O(\kappa^2 m^2)$.

Proof: The proof of the following statements clearly 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) unchanged.
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 $\sum_{u \in \aleph(v)} f^{\theta}_{vu} \geq z(v) > 0$. Hence, by constraints (1) and (2), the total of incoming and generated flows $\sum_{u \in \aleph'(v)} f^{\theta}_{uv} > 0$. 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 $f^{\theta}_{v_i v_{i+1}} = y$ for $i = 1, 2, \cdots, l-1$, $f^{\theta}_{v_l v_1} = y$, and all other flow values

Algorithm Extract-Embeddings: Finding an equivalent solution of *Embedding-Edge-LP* from a feasible solution of *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^{θ}_{uv} $\forall \theta \in \Gamma$, $\forall uv \in E$, and f^{θ}_{uu} $\forall \theta \in \Gamma$, $\forall u \in V$.

output: A subset $\mathcal{B}' \subseteq \mathcal{B}$ and flows $\{x(B)|B \in \mathcal{B}'\}$ with $\sum_{B\in\mathcal{B}'} x(B) = \lambda$ so that these together with $x(B) = 0$ for $B \in \mathcal{B} \setminus \mathcal{B}'$ is a solution to *Embedding-Edge LP*.

1 Initialize $\lambda' = 0$
2 **while** $\lambda' \neq \lambda$ **do**
3 $z' := \lambda$;
4 $B(\theta_{|\Gamma|}) := t$; `// Start defining a new embedding` B
5 $B(\theta_i) = \emptyset$ for $i = 1$ to $|\Gamma| - 1$;
6 **for** $i := |\Gamma|$ **to** 1 **do**
7 $v := \mathsf{end}(B(\theta_i))$;
8 $z(v) = z'$;
9 $u :=$ an element in $\aleph'(v)$ such that $f^{\theta_i}_{uv} > 0$;
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 $y := \min_{u'v' \in \{uv\}\cup P} \left(f^{\theta_i}_{u'v'}\right)$;
14 $f^{\theta}_{u'v'} := f^{\theta}_{u'v'} - y \ \forall u'v' \in \{uv\} \cup P$
15 **end**
16 **else**
17 $z(u) := \min\left(z(v), f^{\theta_i}_{uv}\right)$;
18 **end**
19 **if** $u \neq v$ **then**
20 Prefix u in $B(\theta_i)$;
21 $v := u$;
22 Break and 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 $f^{\theta}_{u'v'} := f^{\theta}_{u'v'} - x(B)$ $\forall \theta \in \Gamma$ and $\forall u'v' \in B(\theta)$;
32 $f^{\theta}_{v'v'} := f^{\theta}_{v'v'} - x(B)$ $\forall \theta \in \Gamma$ and $v' = \mathsf{start}(B(\theta))$;
33 **end**

are equal to 0. Hence, 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 $u = \mathsf{start}(B(\theta))$, 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 $O(m|\Gamma|) = O(m\kappa)$ 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.

Since $|\{f^{\theta}_{uv}|uv \in E, \theta \in \Gamma\}|$ is in $O(\kappa m)$, the total number of cycles traversed and removed is in $O(\kappa m)$. Similarly, the total number of embeddings that the algorithm finds is in $O(\kappa m)$. Also, one can check that the time taken by the algorithm between consecutive removals of cycles or embeddings is in $O(\kappa m)$. Thus, the Extract-Embeddings algorithm has the overall complexity in $O(\kappa^2 m^2)$. ■

The polynomial time-complexity implies that only polynomial number of embeddings will be assigned with non-zero flows by this algorithm. This is compatible with the well known fact that even in traditional multi-commodity flow problems, the solution of the corresponding node-arc LP gives non-zero flows only on polynomial number of paths.

IV. Faster near-optimal solution and min-cost embedding

The *Node-Arc LP* and the *Extract-Embeddings* algorithm to find an optimal solution of the *Embedding-Edge LP* has polynomial-time complexity. We now give the dual of our *Embedding-Edge LP* and present a faster primal-dual algorithm to compute a near-optimal solution, i.e., a solution that achieves at least $(1 - \epsilon)$ fraction of the computing capacity $C(\mathcal{N}, \mathcal{G})$ for any chosen $\epsilon > 0$. This algorithm needs a subroutine which finds a 'minimum weight embedding' of the computation tree in the network for given edge-weights. We present an efficient algorithm for this purpose. This algorithm is of independent interest, for instance, for computing functions over a network with power-limited, but with infinite-bandwidth, links.

We begin by recalling that *Embedding-Edge LP* is a fractional packing problem. For multi-commodity flow packing problems, Garg and Konemann [4] gave a fast primal-dual algorithm to find a near-optimal solution that is within $(1-\epsilon)$ of the optimal packing. Based on this technique, we present a fast algorithm to obtain an $(1 - \epsilon)$-optimal solution to the *Embedding-Edge LP*. A key requirement of the algorithm of [4] is an oracle subroutine that finds the shortest paths between the source-terminal pairs. We too will use an oracle subroutine but our oracle needs to find the minimum weight embedding; we call it *Min-Cost-Embedding*(L). *Min-Cost-Embedding*(L) finds the minimum weighting embedding of $\mathcal{G}$ in a weighted graph $\mathcal{N}$ with weights L.

We first write the dual of the *Embedding-Edge LP*. The dual has the variables $L = \{l(e)\}_{e\in E}$ corresponding to the capacity constraints in the primal. The dual LP is given as follows.

Dual of Embedding-Edge LP: Minimize $D(L) = \sum_{e\in E} c(e)l(e)$ subject to

1. Constraints corresponding to each $x(B)$ in primal:

$$\sum_{e\in B} r_B(e)l(e) \geq 1,\ \forall B \in \mathcal{B}$$

2. Non-negativity constraints:

$$l(e) \geq 0,\ \forall e \in E$$

We define the weight of an embedding B as

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

Following the method of [4], it can be checked that the dual LP is equivalent to finding $\min_L \frac{D(L)}{\alpha_L}$, where

$$\alpha_l = \min_B w_l(B)$$

is the cost of the minimum cost embedding for L.

For a packing LP of the form

$$\max\left\{a^T x | Ax \leq b, x \geq 0\right\}$$

and its dual LP of the form

$$\min\left\{b^T y | A^T y \geq a, y \geq 0\right\},$$

the shortest path is defined as $\sum_i A(i,j)y(i)/a(j)$ [4]. It can be seen that for the *Embedding-Edge LP*, the 'shortest path' corresponds to the embedding with minimum weight, i.e., $\arg\min_B w_L(B)$. Algorithm FANO below gives the instance of the primal-dual algorithm for the *Embedding-Edge LP*.

Algorithm FANO: Fast Algorithm for finding near optimal x and λ

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 ϵ

output: Primal solution $\{x(B), B \in \mathcal{B}\}$ s.t. $\sum_{B\in\mathcal{B}} x(B) \geq (1-\epsilon)C(\mathcal{N}, \mathcal{G})$

Initialize $l(e) := \delta/c(e), \forall e \in E, x(B) := 0, \forall B \in \mathcal{B}$;
while $D(l) < 1$ **do**
  $B^* :=$ *Min-Cost-Embedding*(L) ;
  `// Min-Cost-Embedding(L) outputs` $\arg\min_B w_L(B)$
  $e^* :=$ edge in B^* 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
$x(B) := x(B)/\log_{1+\epsilon}\frac{1+\epsilon}{\delta}, \forall B$;

We now describe, and then provide below, the subroutine Min-Cost-Embedding*(L)* that finds a minimum weight embedding of $\mathcal{G}$ on $\mathcal{N}$ with a given length/cost function L. For each edge θ_i, starting from θ_1, the algorithm finds a way to compute θ_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 θ_i. If θ_i is computed at that node itself then the predecessor node is itself. This is done for each θ_i by a technique similar to Dijkstra's shortest path algorithm. Computing θ_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 θ_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 θ_i with the cost of computing $\Phi_\uparrow(\theta_i)$ at the same node. With this initialization, the same principle of greedy node selection and cost update as in Dijkstra's algorithm is used to find the optimal way of obtaining θ_i at all the nodes. Finally, the optimal embedding is obtained by backtracking the predecessors. Starting from t, we backtrack using predecessors from which $\theta_{|\Gamma|}$ is obtained, till we hit a node whose predecessor is itself. This node is the start node of $B(\theta_{|\Gamma|})$ and the end node of $B(\eta)$ for all $\eta \in \Phi_\uparrow(\theta_{|\Gamma|})$. The complete embedding is obtained by continuing this process for each θ_i in the reverse topological order. This is described in Procedure *Min-Cost-Embedding*.

Procedure Min-Cost-Embedding(L)

input : Network graph $\mathcal{N} = (V, E)$, Length function L, set of source nodes S, terminal node t, computation tree $\mathcal{G} = (\Omega, \Gamma)$.

output: Embedding B^* with minimum weight under L

for $i = 1$ **to** $|\Gamma|$ **do**
  if $i \in \{1, 2, \ldots, \kappa\}$ **then**
    $\omega_u(\theta_i) := \infty, \forall u \in V - \{s_i\}$;
    $\omega_{s_i}(\theta_i) := 0$ and $\sigma_{s_i}(\theta_i) := s_i$;
  end
  else
    $\omega_u(\theta_i) := \sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_u(\eta), \forall u \in V$;
    $\sigma_u(\theta_i) := u,\ \forall u \in V$;
  end
  $\Psi := \emptyset$; $\bar{\Psi} := V$;
  while $|\Psi| < n$ **do**
    $v := \arg\min_{u\in\bar{\Psi}} \omega_u(\theta_i)$;
    $\Psi := \Psi \cup \{v\}$;
    $\bar{\Psi} := \Psi - \{v\}$;
    foreach $u \in \aleph(v)$ **do**
      if $\omega_v(\theta_i) + l(uv) < \omega_u(\theta_i)$ **then**
        $\omega_u(\theta_i) := \omega_v(\theta_i) + l(uv)$;
        $\sigma_u(\theta_i) := v$;
      end
    end
  end
end
$B^*(\theta_{|\Gamma|}) := t$;
for $i = |\Gamma|$ **to** 1 **do**
  $u := \mathsf{end}(B^*(\theta_i))$;
  while $\sigma_u(\theta_i) \neq u$ **do**
    Prefix $\sigma_u(\theta_i)$ to $B^*(i)$;
    $u := \sigma_u(\theta_i)$;
  end
  $B^*(\eta) := u\ \forall \eta \in \Phi_\uparrow(\theta_i)$;
end

Theorem *3:* Min-Cost-Embedding*(L)* is correct and has time complexity $O(\kappa(m + n\log n))$.

Proof: To prove the correctness, it is sufficient to show that, during each phase i, the algorithm computes optimal values for $\omega_u(\theta_i)$ and $\sigma_u(\theta_i)$, for each node u in $\mathcal{N}$. We prove this by induction on the pair $(i, |\Psi|)$ according to the lexicographic ordering. For $i \in \{1, \ldots, \kappa\}$ and for all $|\Psi|$, this follows from the correctness of Dijkstra's algorithm. Now, assuming the optimality of $\omega_u(\theta_i)$ and $\sigma_u(\theta_i)$ till all iterations before $(i, |\Psi|)$, we prove the statement for $(i, |\Psi|)$. Suppose v is the element added to Ψ in the current iteration. We consider two cases:

- Case 1: $\Psi = \{v\}$: The cost of computing (and not receiving from another node) θ_i at any node u is $\sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_u(\eta)$. The algorithm chooses v which has the minimum $\sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_u(\eta)$ among all nodes $u \in V$ and assigns $\omega_v(\theta_i) = \sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_v(\eta)$ and $\sigma_v(\theta_i) = v$. If these are not optimal, then it must be more efficient for v to receive θ_i which is computed at some other node u. But that implies $\sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_u(\eta) < \sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_v(\eta)$, which is a contradiction to the choice of v.
- Case 2: $\{v\} \subsetneq \Psi$: Suppose there is a more efficient way of receiving θ_i at v than from the node selected as $\sigma_v(\theta_i)$ and that is to compute θ_i at a node u and receive it along a path $P_{u,v}$. Let the corresponding cost be $\omega'_v(\theta_i)$. First, if $u \in \bar{\Psi}$, then the present cost $\left(\leq \sum_{\eta\in\Phi_\uparrow(\theta_i)} \omega_u(\eta)\right)$ at u is less than the present value of $\omega_v(\theta_i)$, which is a contradiction to the choice of v. Thus $u \in \Psi$. Let u' be the last node in $P_{u,v}$ from Ψ, and v' be the first node in $P_{u,v}$ from $\bar{\Psi}$. Then $\omega'_v(\theta_i) \geq \omega_{u'}(\theta_i) + l(u'v') \geq \omega_{v'}(\theta_i) \geq \omega_v(\theta_i)$— a contradiction. Here the first inequality follows since $u' \in \Psi$. The second inequality follows from the update rule followed during the inclusion of u' in Ψ. The last inequality follows from the choice of v.

This completes the proof of correctness. To obtain the complexity, consider the first **for** loop in *Min-Cost-Embedding*(L). Each iteration of this loop is the same as Dijkstra's algorithm except for the initialization. Thus, the for loop, excluding the initialization step, can be run in $O(m + n\log n)$ time using Fibonacci heap implementation. The initialization step requires $O(n|\Phi_\uparrow(\theta_i)|)$ time for each iteration. The second **for** loop has $O(n\kappa)$ complexity. So the overall algorithm takes $O(\kappa(m + n\log n))$ time.

The number of iterations in the primal-dual algorithm is of the order $O(\epsilon^{-1} m \log_{1+\epsilon}(m))$. Thus the overall complexity of the algorithm is $O\left(\epsilon^{-1}\kappa m(m + n\log n)\log_{1+\epsilon}(m)\right)$.

A. Distributed implementation:

One of the distinctive feature of the above primal-dual algorithm is its amenability to distributed implementation. First, consider distributed implementation of the oracle. As noted above, an execution of *Min-Cost-Embedding*(L) has two stages—(1) initialization and execution of a sequence of shortest path algorithm, and (2) obtaining the corresponding min-weight embedding. The initialization of the shortest path algorithm at each node depends only on the values from previous iterations at that node and the knowledge of computation graph.

Efficient distributed implementation for finding shortest paths, wherein information is shared only among neighbors, are well studied [24]. Further, the knowledge of the min-weight embedding is implicit in the network in the form of $\sigma_u(\theta)$ at each node u, for each $\theta \in \Gamma$. For example, at the destination node t, if $\sigma_t\left(\theta_{|\Gamma|}\right) \neq t$, node t needs to communicate and obtain $\theta_{|\Gamma|}$ from its neighbor $u = \sigma_t\left(\theta_{|\Gamma|}\right)$. Now, if $\sigma_u(\theta) = u$, node u needs to compute θ by obtaining $\Phi_\uparrow(\theta)$ from nodes $\sigma_u(\eta)$ for all $\eta \in \Phi_\uparrow(\theta)$, and so on. Thus, the distributed version of the algorithm would be of the following form. Periodically, each node computes weights of its incident edges as a function of their capacity and total flow through them. Then, the nodes run the sequence of distributed shortest path algorithm to obtain $\sigma_u(\theta)$ for each θ. A fraction of the flow generated at the sources is then transmitted through the resulting embedding using the knowledge of $\sigma_u(\theta)$. Rigorous development of such an algorithm to obtain fault-tolerant and adaptive distributed implementation would be yet another interesting avenue for further research.

V. Extensions and Open Problems

1. Multiple trees for the same function: A function may have many possible computation trees. For example, the well-investigated [17] '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 thus be 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. Also, *Min-Cost-Embedding*(L) algorithm can be made more efficient by running iterations for each function rather than each edge. The initialization of $\omega_u(\theta)$ changes correspondingly, to take into account all possible ways of computing that function. Rest of the algorithm remains the same.

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. Both *Node-Arc-LP* and the primal-dual algorithm find 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 in turn equivalent to the fractional Steiner tree packing problem which is known to be NP-complete. Note, however, that our technique suggests a suboptimal technique of considering only a subset of all possible computation trees, which would result in suboptimal 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.

For the multicast problem, and consequently for the function 'sum', the oracle finds a minimum weight Steiner tree, the well-known NP-hard problem. Almost optimal (but not near-optimal, that is, not $(1-\epsilon)$-optimal for any given ϵ) polynomial complexity algorithms are known (see [25] and citations therein) for finding a minimum weight Steiner tree. This can also be used to find almost optimal solution to the multicast, and hence the 'sum', in polynomial complexity [25].

2. Multiple functions and multiple terminals: Suppose the network has multiple terminals $t_1, t_2, \ldots, t_\gamma$ wanting functions $\Theta_1(X^{(1)}), \Theta_2(X^{(2)}), \ldots, \Theta_\gamma(X^{(\gamma)})$ respectively. Here $X^{(i)}$ is the data generated by a set of sources $S^{(i)}$. The sets $S^{(i)}; i = 1, 2, \ldots, \gamma$ are assumed to be pairwise disjoint. For each function Θ_i, a computation tree $\mathcal{G}_i$ is given. Let us consider the problem of communicating the functions to the respective terminals at rates $\lambda_1, \lambda_2, \ldots, \lambda_\gamma$. The problem is to determine the achievable rate region which is defined as the set of $\mathbf{r} = (\lambda_1, \lambda_2, \ldots, \lambda_\gamma)$ for which a protocol exists for transmission of the functions at these rates simultaneously. This region can be approximately found by solving either of the following problems.

(i) For any given non-negative weights $\alpha_1, \alpha_2, \ldots, \alpha_\gamma$, what is the maximum achievable weighted sum-rate $\sum_{i=1}^{\gamma} \alpha_i \lambda_i$?

For this problem, we consider embeddings of the computation trees $\mathcal{G}_i$ into the network for each terminal t_i. Let $\mathcal{B}_i$ denote the set of all embeddings of $\mathcal{G}_i$. Then the *Embedding-Edge LP* for this problem is to maximize $\sum_{i=1}^{\gamma} \alpha_i \sum_{B \in \mathcal{B}_i} x(B)$. The constraints are the same as before with $\mathcal{B}$ defined by $\mathcal{B} = \cup_i \mathcal{B}_i$. The weight of an embedding $B \in \mathcal{B}$ under a weight function L is defined as $\alpha_i w_L(B)$ if $B \in \mathcal{B}_i$. The new *Min-Cost-Embedding*(L) algorithm finds an optimal embedding for each $\mathcal{G}_i$ and chooses the one with minimum weight. This can be used in the same primal-dual algorithm to find a near-optimal solution. It is also easy to obtain a *Node-Arc LP* for this problem by minor modifications to that for a single function computation at a single terminal.

(ii) For any non-negative demands $\alpha_1, \alpha_2, \ldots, \alpha_\gamma$, what is the maximum λ for which the rates $\lambda\alpha_1, \lambda\alpha_2, \ldots, \lambda\alpha_\gamma$ are concurrently achievable?

Here, we define an embedding to be a tuple $B = (B_1, B_2, \ldots, B_\gamma)$, where $B_i \in \mathcal{B}_i$ is an embedding of the computation tree $\mathcal{G}_i$. The *Embedding-Edge LP* for this problem is the same as that for the single terminal problem with $r_B(e)$ defined as $r_B(e) = \sum_{i=1}^{\gamma} \alpha_i |\{\theta \in \Gamma_i | e \text{ is a part of } B_i(\theta)\}|$ and $\mathcal{B} = \mathcal{B}_1 \times \mathcal{B}_2 \times \ldots \times \mathcal{B}_\gamma$. The weight of an embedding B under a weight function L is defined as $\sum_{i=1}^{\gamma} \alpha_i w_L(B_i)$. The new *Min-Cost-Embedding*(L) algorithm finds an optimal embedding B by separately finding optimal embeddings B_i for each $\mathcal{G}_i$. This can be used in the same primal-dual algorithm to find a near-optimal solution. Again, we can easily obtain a *Node-Arc LP* by minor modification to that for a single function computation at a single terminal.

3. **Computing with a specified precision:** In practice, the source data may be real-valued, and communicating such a data requires infinite capacity. In such applications, it is common to require a quantized value of the function at the terminal with a desired precision. This may, in turn, be achieved by quantizing various data types with pre-decided precisions and thus different data type may require different number of bits to represent them. Suppose the data type denoted by θ is represented using $b(\theta)$ bits. Then the *Embedding-Edge LP* and its dual for this problem are the same as before except that the definition of $r_B(e)$ is changed to $r_B(e) = \sum_{\theta \in \Gamma: e \text{ is a part of } B(\theta)} b(\theta)$. In the *Node-Arc LP*, the capacity constraints are changed to

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

In the *Min-Cost-Embedding*(L) algorithm, $l(uv)$ is replaced by $l(uv)b(\theta_i)$ inside the **foreach** loop.

4. **Energy limited sensors:** Suppose, instead of capacity constraints on the links, each node $u \in V$ has a total energy $E(u)$. Each transmission and reception of θ require the energy $E_{T,\theta}$ and $E_{R,\theta}$ respectively. Generation of one symbol of θ or computation of one symbol of θ from $\Phi_{\uparrow}(\theta)$ requires the energy $E_{C,\theta}$. The objective is to compute the function at the terminal maximum number of times with the given total node energy at each node.

For an embedding B, if $B(\theta) = v_1, v_2, \cdots, v_l$, then $tr(B(\theta)) = \{v_1, v_2, \cdots, v_{l-1}\}$ denotes the transmitting nodes, and $rx(B(\theta)) = \{v_2, v_3, \cdots, v_l\}$ denotes the receiving nodes of θ. If $l = 1$, then $tr(B(\theta)) = rx(B(\theta)) = \emptyset$. For B, the energy load on the node u is given by

$$E_B(u) = \sum_{\theta: \mathsf{start}(B(\theta)) = u} E_{C,\theta} + \sum_{\theta: u \in tx(B(\theta))} E_{T,\theta} + \sum_{\theta: u \in rx(B(\theta))} E_{R,\theta}.$$

The capacity constraint in the *Embedding-Edge LP* is replaced by the energy constraint on the nodes

$$\sum_{B \in \mathcal{B}} x(B) E_B(u) \leq E(u) \ \forall u \in V,$$

where an empty sum is defined to be 0. The dual of the *Embedding-Edge LP* is: Minimize $D(L) = \sum_{u \in V} E(u) l(u)$ subject to

1. Constraints corresponding to each $x(B)$ in primal:

$$\sum_{u \in B} E_B(u) l(u) \geq 1, \ \forall B \tag{3}$$

2. Non-negativity constraints:

$$l(u) \geq 0, \ \forall u \in V. \tag{4}$$

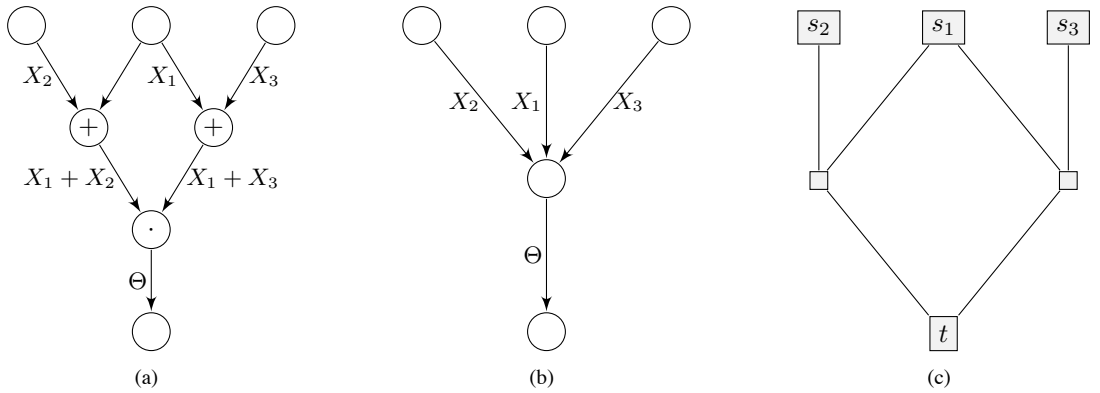


Fig. 5. Computing $\Theta = (X_1 + X_2)(X_1 + X_3)$ over a network. (a) A DAG computation schema. (b) The trivial star computation tree. (c) A network for computing Θ.

The weight or cost of an embedding is defined as

$$w_L(B) = \sum_{u \in B} E_B(u) l(u).$$

Min-Cost-Embedding(L) is modified in the weight initialization and weight update. The weight initialization is done as $\omega_{s_i}(\theta_i) := E_{C,\theta_i}$ for source data and $\omega_u(\theta_i) := E_{C,\theta_i} + \sum_{\eta \in \Phi_\uparrow(\theta_i)} \omega_u(\eta)$ for other data. The weight update at u is now done as $\omega_u(\theta_i) := \omega_v(\theta_i) + E_{T,\theta_i} + E_{R,\theta_i}$ if $\omega_v(\theta_i) + E_{T,\theta_i} + E_{R,\theta_i} < \omega_u(\theta_i)$. After suitable modification, the primal-dual algorithm with the modified *Min-Cost-Embedding*(L) algorithm finds a near-optimal solution.

In the *Node-Arc LP*, the capacity constraints are replaced by energy constraints at the nodes:

$$\sum_{\theta \in \Gamma} f^\theta_{uu} E_{C,\theta} + \sum_{\theta \in \Gamma} \sum_{v \in \aleph(u)} (f^\theta_{uv} E_{T,\theta} + f^\theta_{vu} E_{R,\theta}) \leq E(u) \quad \forall u \in V.$$

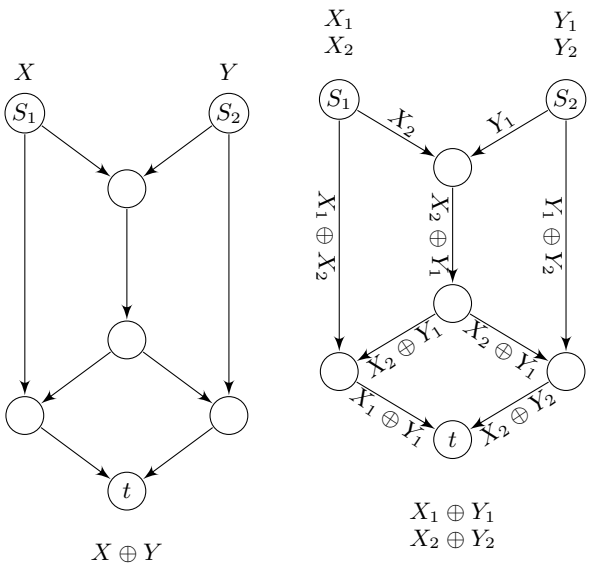
Since the flow on each embedding is required to be integer in this set-up, the *Embedding-Edge-LP* and *Node-Arc-LP* are both integer programs. Since solving integer programs is in general difficult, the exact solution of this problem is difficult. However, the LP relaxation of the problem can be solved efficiently. In a practical application, if the initial energy levels of the nodes are large (that is, the solution involves large number computations on the used embeddings), then the simple truncation of the solution of the LP relaxation will give an acceptable solution. Quantifying the gap of this solution from the optimal solution is an interesting open problem, and is outside the scope of this paper.

5. **Computation schema represented by a directed acyclic graph (DAG):** For some functions, it may be more efficient to perform the computation in a sequence of computations which form a directed acyclic graph instead of a directed tree. For example, consider the function $\Theta(X_1, X_2, X_3) = (X_1 + X_2)(X_1 + X_3)$. A natural efficient way of computing is represented by the computation schema shown in Fig. 5(a). The alternative trivial 'star' computation tree, which represents bringing all the data X_1, X_2, X_3 to one node for computation is also shown in Fig. 5(b). Though the optimal computation rate according to the star computation tree can be found using our algorithms, using the DAG computation schema may give a better rate than the best achieved using the star computation tree. An illustrative example network where this happens is shown in Fig. 5(c). Here each edge has a capacity of one symbol per use. Clearly using the natural embedding of the DAG computation schema in this network provides a rate of 1 computation per unit time. But, it can be seen that the best rate achieved (using equal time-sharing of two embeddings) using the star computation schema is $2/3$. However, our solution techniques are not adequate in general for providing the most efficient solution using a DAG computation schema. Efficient solution of the problem with DAG computation schema remains an interesting open problem.

VI. 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, for instance, 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. The sum function discussed in Sec. V is one such function representable by a hypergraph.

Our computation framework does not allow block coding, i.e., coding across different realizations of the data. Such coding has been used in the information theory and network coding literature. Block coding can, in general, offer better computation rate. For example, consider the directed butterfly network as shown in Fig. 6 with two binary source nodes (with source processes denoted by X and Y) and a terminal node with a XOR target function $\Theta(X, Y) = X \oplus Y$. It can be checked that the maximum rate achievable by routing-like schemes, i.e., without using inter-realization coding, is 1.5. On the other hand, the scheme shown in Fig. 6(b) using inter-realization coding achieves a rate of 2. However, for more general functions, 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 [20]. We expect that similar results will hold for function computation over undirected networks.



(a) The butterfly network. Each edge has capacity 1 bit/use

(b) A rate-2 solution using cross-realization coding

Fig. 6. The butterfly network with XOR target function $\Theta(X, Y) = X \oplus Y$

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

VII. Acknowledgement

The authors would like to thank A. Diwan for fruitful discussions, and the reviewers for constructive feedback that has improved the presentation of this work.

Appendix A
Proof outline of Theorem 1

Achievability: We will show that for any $\{x(B)|B \in \mathcal{B}\}$ that satisfies the constraints of the *Embedding-Edge-LP* , the rate $\sum_{B\in\mathcal{B}} x(B)$ is $(\mathcal{N}, \mathcal{G})$-achievable. Let $\epsilon > 0$ be any small positive real number. Since the rational numbers are dense, we can find a set of rational flows $\{x'(B)|B \in \mathcal{B}\}$ such that $\sum_{B\in\mathcal{B}} x'(B) \geq \sum_{B\in\mathcal{B}} x(B) - \epsilon$. Let η be the least common multiple of the denominators of $\{x'(B)|B \in \mathcal{B}\}$. We take $K = \eta \sum_{B\in\mathcal{B}} x'(B)$. Let $N_e = \eta \sum_{B\in\mathcal{B}} r_B(e)x'(B)$ for all $e \in E$. Let us fix an order $B_1, B_2, \ldots, B_{|\mathcal{B}|}$ of the embeddings. Let $L(B) = \sum_{e\in E} r_B(e)$ denote the number of edges taking part in the embedding B. We will construct a routing-computing scheme with the following features:

1. It communicates $K = \eta \sum_{B\in\mathcal{B}} x'(B)$ realizations of the function in one session. Out of them, $\eta x'(B)$ realizations are communicated using embedding B.

2. It uses any edge e to communicate $N_e = \eta \sum_{B\in\mathcal{B}} r_B(e)x'(B)$ symbols.

3. It has $L = \eta \sum_{B\in\mathcal{B}} L(B)x'(B)+\eta \sum_{B\in\mathcal{B}} x'(B)(|\Omega|-\kappa)$ events. It has $\eta \sum_{B\in\mathcal{B}} L(B)x'(B)$ communication events, and $\eta \sum_{B\in\mathcal{B}} x'(B)(|\Omega| - \kappa)$ computation events[3].

A routing-computing scheme with the above parameters clearly satisfies $N_e(\sum_{B\in\mathcal{B}} x(B) - \epsilon)$ $\leq N_e \sum_{B\in\mathcal{B}} x'(B) \leq Kc(e)$, $\forall e \in E$, and thus guarantees the achievability of the computing rate $\sum_{B\in\mathcal{B}} x(B)$. We now describe the scheme. Let

$$\psi_B : [1 : L(B) + (|\Omega| - \kappa)] \longrightarrow V \cup (V \times V)$$

be a topological ordering of the (non-source) computation nodes and the edges of the embedding B obtained by considering a topological ordering of the non-source nodes and the edges of $\mathcal{G}$ and then replacing each edge θ by the edges in $B(\theta)$ in topological order. Further, let

$$\phi_B : [1 : L(B) + (|\Omega| - \kappa)] \longrightarrow \Gamma$$

be such that $\phi_B(i)$ is the data that is computed (if $\psi_B(i) \in V$) or carried (if $\psi_B(i) \in V \times V$) by $\psi_B(i)$ in the embedding B.

We define the sets $\mathcal{D}_{v,l} \subseteq \mathcal{D}$; $\forall v \in V$ and $\forall l \in [1, L + 1]$ below in an algorithmic fashion.

1. For $1 \leq i \leq \kappa$, $\mathcal{D}_{s_i,1} = \{(\theta_i, k)|1 \leq k \leq K\}$. For all $v \in V \setminus \{s_i|1 \leq i \leq \kappa\}$, $\mathcal{D}_{v,1} = \emptyset$.
2. For each $i = 1, 2, \cdots, |\mathcal{B}|$,
   For each $j = 1, 2, \cdots, \eta x'(B_i)$,
   Let $k = \sum_{\nu=1}^{i-1} \eta x'(B_\nu) + j$. We now describe the events for the k-th realization of data.
   For each $n = 1, 2, \cdots, L(B_i) + (|\Omega| - \kappa)$,
   Let $l = \eta \sum_{\nu=1}^{i-1} (L(B_\nu)x'(B_\nu) + x'(B_\nu)(|\Omega| - \kappa)) + (j - 1)(L(B_i) + (|\Omega| - \kappa)) + n$

   (i) If $v = \psi_{B_i}(n) \in V$, then the l-th event is a computation of $\theta = \phi_{B_i}(n)$ at v. The condition $\Phi_\uparrow(\theta) \subseteq \mathcal{D}_{v,l}(k)$ holds because of the topological order of the events. The data-sets $\mathcal{D}_{u,l+1}$ are defined in terms of $cD_{u,l}$ as given in condition 2(i) of the definition of a routing-computing scheme. That is, $\mathcal{D}_{v,l+1} = \{(\theta, k)\} \cup \mathcal{D}_{v,l} \setminus \{(\eta, k)|\eta \in \Phi_\uparrow(\theta)\}$. For all $u \in V \setminus \{v\}$, $\mathcal{D}_{u,l+1} = \mathcal{D}_{u,l}$.

   (ii) If $(u, v) = \psi_{B_i}(n) \in (V \times V)$ and $\theta = \phi_{B_i}(n)$, then the l-th event is a communication of $\theta(\mathbf{X}(k))$ from u to v over the edge (u, v). $(\phi_{B_i}(n), k) \in \mathcal{D}_{u,l}$ holds because of the topological order of the events. The data-sets $\mathcal{D}_{w,l+1}$ are defined in terms of $cD_{w,l}$ as given in condition 2(ii) of the definition of a routing-computing scheme. That is, $\mathcal{D}_{u,l+1} = \mathcal{D}_{u,l} \setminus \{(\theta, k)\}$, and $\mathcal{D}_{v,l+1} = \mathcal{D}_{v,l} \cup \{(\theta, k)\}$. For any $w \neq u, v$, $\mathcal{D}_{w,l+1} = \mathcal{D}_{w,l}$.

   Again, by the topological order of the defined events, it is easy to check that the conditions: $\mathcal{D}_{t,L+1} = \{\Theta(\mathbf{X}(k))|1 \leq k \leq K\}$, $\mathcal{D}_{v,L+1} = \emptyset\ \forall v \neq t$, and for all $e \in E$,

   $$|\{l \in [1, L]|E_l \text{ is a communication over } e\}| = N_e$$

   are satisfied.

Converse: It is sufficient to prove that, for any given $(\{N_e|e \in E\}, K)$ routing-computing scheme for $(\mathcal{N}, \mathcal{G})$ satisfying

$$N_e\lambda \leq c(e)K,\ \forall e \in E, \tag{5}$$

there exists a packing $\{x(B)|B \in \mathcal{B}\}$ satisfying the constraints of the *Embedding-Edge-LP* such that $\sum_{B\in\mathcal{B}} x(B) = \lambda$.

[3]Note that $(|\Omega| - \kappa)$ is the number of non-source nodes in the computation graph – each computation of Θ requires these many computation steps.

Since the given $(\{N_e|e \in E\}, K)$ scheme computes the function $\Theta(\mathbf{X}(k))$ for $k = 1, 2, \ldots, K$, for each k, the computation uses an embedding $B^{(k)} \in \mathcal{B}$. In particular, for each $e \in E$, the k-th computation requires communication of $r_{B^{(k)}}(e)$ symbols over e. So, we have

$$\sum_{k=1}^{K} r_{B^{(k)}}(e) = N_e \tag{6}$$

for all $e \in E$. Now, let us define

$$x(B) = \frac{\lambda|\{k \in [1, K]|B^{(k)} = B\}|}{K} \tag{7}$$

for all $B \in \mathcal{B}$. By definition, these are non-negative, and $\sum_{B\in\mathcal{B}} x(B) = \lambda$. Eq. (6) can be rewritten as

$$\begin{aligned}
&\sum_{B\in\mathcal{B}} |\{k \in [1, K]|B^{(k)} = B\}| r_B(e) = N_e \\
\Rightarrow &\sum_{B\in\mathcal{B}} Kx(B) r_B(e) = \lambda N_e \leq Kc(e) \quad \text{(using (7) and (5))} \\
\Rightarrow &\sum_{B\in\mathcal{B}} x(B) r_B(e) \leq c(e)
\end{aligned}$$

So, $\{x(B)|B \in \mathcal{B}\}$ satisfies the conditions of the *Embedding-Edge-LP* . Thus $\{x(B)|B \in \mathcal{B}\}$ provides a solution of *Embedding-Edge-LP* with $\sum_{B\in\mathcal{B}} x(B) = \lambda$.

Appendix B
Converting Timeshare Allocation into a Schedule

Appendix A shows that there exists a routing-computing scheme achieving a total computation rate arbitrarily close to any feasible solution of *Embedding-Edge-LP* . Such a routing-computing scheme can be implemented in a 'pipelined' fashion provided there is no limitation on the latency of computation and the available memory at each node. Such an implementation will ensure that all the nodes (respectively links) are used simulteneously for computing (respectively communicating) albeit for different realizations of data. This requires careful design of a scheduling protocol to ensure that only the data from the same realization are used for any computation at any node. In this section, we provide such a scheduling protocol to compute at a rate which is arbitrarily close to a solution of *Embedding-Edge-LP* .

Let $\{x(B)|B \in \mathcal{B}\}$ be a solution of *Embedding-Edge-LP* with rate $\lambda = \sum_{B\in\mathcal{B}} x(B)$. For any $\epsilon > 0$, we now outline a communication and computation protocol designed to receive the function at the terminal at a rate that is greater than $\lambda - \epsilon$. First, the flow values $\{x(B)|B \in \mathcal{B}\}$ are rounded to lower rational numbers $\{x'(B)|B \in \mathcal{B}\}$ so that the total flow $\sum_{B\in\mathcal{B}} x'(B) > \lambda - \epsilon$. All these flows are then multiplied by the least common multiple N of the denominators of the flows $x'(B); B \in \mathcal{B}$. Let the resulting values be $n(B); B \in \mathcal{B}$. Let $K = \sum_{B\in\mathcal{B}} n(B)$. Clearly, $K > (\lambda-\epsilon)N$. Let us fix an order in the embeddings $B_1, B_2, \ldots, B_{|\mathcal{B}|}$. The protocol consists of computation at the nodes and communication across the links in a block/frame of N consecutive uses of the network. In each frame, a link e can carry upto a total of $Nc(e)$ symbols in both directions. Our protocol will require sending integer number of symbols in N uses of e in each direction. We assume that this is possible as long as the total number of symbols transmitted in both directions is at most $Nc(e)$. We assume that computation at nodes is done instantaneously, and a frame sent across a link is available at the receiving node at the end of the frame. The receiving node can forward the data on another edge in the next frame or use it to compute something else for transmission in the next or later frames.

In our protocol, the data stream generated at each source is divided into blocks of K symbols, and the terminal computes K number of corresponding function values in each frame. Out of the K computations, the first $n(B_1)$ are carried out using the embedding B_1, the next $n(B_2)$ are carried out using the embedding B_2, and so on. In each direction on each link, the transmissions corresponding to different embeddings are ordered in the same order as the embeddings. Further, if uv is in $B(\theta_i)$ as well as $B(\theta_j)$ (assume $i < j$ without loss of generality), then uv carries the data for (B, θ_i) first and then the data for (B, θ_j). Formally, in each frame and in each direction, a link uv in $\mathcal{N}$ carries a subframe, possibly empty, of data for each (B, θ) pair, where $B \in \mathcal{B}, \theta \in \Gamma$. These subframes are transmitted in the lexicographic order on (B, θ). Since the subframes for different (B, θ) may be available at u with different delay, these subframes will not correspond to the same frame of source data. In the following, we explicitly describe the subframes carried by uv in the k-th frame.

Let $\mathbf{y}^k_{B,\theta}$ denote the $n(B)$ symbols of data of type θ corresponding to the $n(B)$ symbols of source data in the k-th frame corresponding to the embedding B. That is, $\mathbf{y}^k_{B_1,\theta}$ denotes the $n(B_1)$ symbols of data of type θ corresponding to the first $n(B_1)$ symbols of source data in the k-th frame, $\mathbf{y}^k_{B_2,\theta}$ denotes the $n(B_2)$ symbols of data of type θ corresponding to the next $n(B_2)$ symbols of source data in the k-th frame, and so on. In each frame, uv carries a subframe of data for each (B, θ) pair. The subframe corresponding to (B, θ) is empty if $uv \notin B(\theta)$. Formally,

$$\mathbf{y}^k_{uv,B,\theta} = \begin{cases} \mathbf{y}^k_{B,\theta} \text{ if } uv \in B(\theta), \\ \emptyset \text{ otherwise.} \end{cases}$$

This subframe corresponds to the k-th block of source data. These subframes may be available at u with variable delay due to variable path lengths from the sources along different embeddings. Let us define the depth or delay $d(u, B, \theta)$ as

$$d(uv, B, \theta) = \begin{cases} \infty \text{ if } uv \notin B(\theta) \\ 0 \text{ if } uv \in B(\theta), u = s_i, \theta = \theta_i \\ 1+\max\{d(wu, B, \eta)|\eta \in \Phi_\uparrow(\theta), wu \in B(\eta)\}\} \\ \quad \text{if } uv \in B(\theta), u = \mathsf{start}(B(\theta)), \\ \qquad\qquad (u, \theta) \neq (s_i, \theta_i) \\ d(wu, B, \theta) + 1 \text{ if } (u, \theta) \neq (s_i, \theta_i), \\ \qquad\qquad wu, uv \in B(\theta). \end{cases} \tag{8}$$

So, the subframe $\mathbf{y}^k_{uv,B,\theta}$, which has $n(B)$ symbols if $uv \in B(\theta)$ and which corresponds to the k-the frame of source data, will be transmitted in the $(k + d(uv, B, \theta))$-th frame on uv. The infinite value for $uv \notin B(\theta)$ indicates that the corresponding data does not flow through uv from u to v.

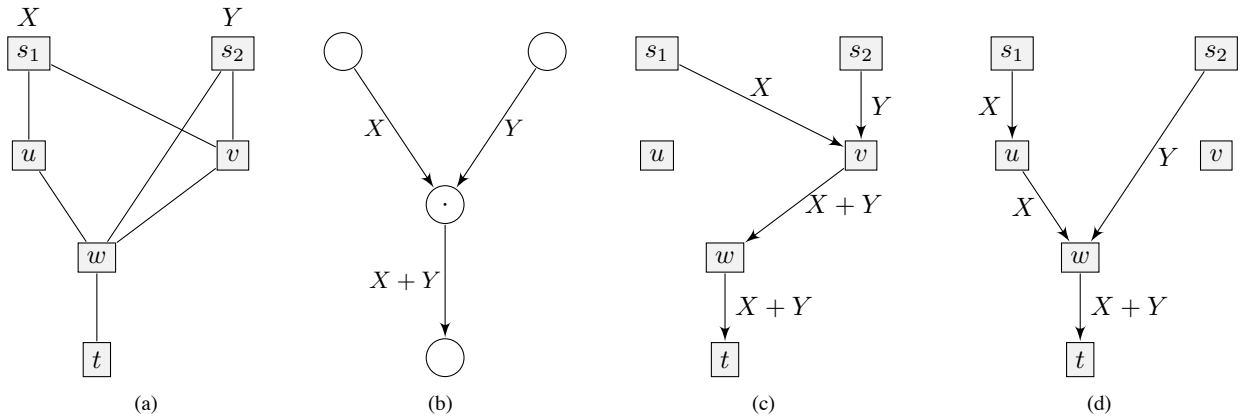


Fig. 7. A network, a computation tree and two embeddings

Example *4:* Consider the network and the computation tree shown in Fig. 7. The edges of the computation tree are labeled by the functions they carry, that is, X, Y, and $X+Y$. For embedding B_1, $d(s_1v, B_1, X) = 0$, $d(s_2v, B_1, Y) = 0$, $d(vw, B_1, X+Y) = 1$, $d(wt, B_1, X+Y) = 2$, and all other delay values are ∞. For embedding B_2, $d(s_1u, B_2, X) = 0$, $d(s_2w, B_2, Y) = 0$, $d(uw, B_2, X) = 1$, $d(wt, B_2, X+Y) = 2$, and all other delay values are ∞.

The data transmitted in the k-th frame from u to v on the link uv, in order of transmission, is thus $\mathbf{y}_{uv,B_1,\theta_1}^{k-d(uv,B_1,\theta_1)}$, $\mathbf{y}_{uv,B_1,\theta_2}^{k-d(uv,B_1,\theta_2)}$, $\cdots$, $\mathbf{y}_{uv,B_1,\theta_{|\Gamma|}}^{k-d(uv,B_1,\theta_{|\Gamma|})}$, $\mathbf{y}_{uv,B_2,\theta_1}^{k-d(uv,B_2,\theta_1)}$, $\mathbf{y}_{uv,B_2,\theta_2}^{k-d(uv,B_2,\theta_2)}$, $\cdots$, $\mathbf{y}_{uv,B_2,\theta_{|\Gamma|}}^{k-d(uv,B_2,\theta_{|\Gamma|})}$, $\cdots$, $\mathbf{y}_{uv,B_{|\mathcal{B}|},\theta_1}^{k-d(uv,B_{|\mathcal{B}|},\theta_1)}$, $\mathbf{y}_{uv,B_{|\mathcal{B}|},\theta_2}^{k-d(uv,B_{|\mathcal{B}|},\theta_2)}$, $\cdots$, $\mathbf{y}_{uv,B_{|\mathcal{B}|},\theta_{|\Gamma|}}^{k-d(uv,B_{|\mathcal{B}|},\theta_{|\Gamma|})}$. It is easy to see that the required flow of function values will be computed on each embedding by this protocol. If the communication starts with the frame number 0 and ends with the M-th frame of source data, then the subframes are empty for $k < d(uv, B_i, \theta_j)$ and for $k > M + d(uv, B_i, \theta_j)$. In particular, a subframe $\mathbf{y}_{uv,B_i,\theta_j}^{k-d(uv,B_i,\theta_j)}$ is empty if $uv \notin B_i(\theta_j)$.

Example *5:* In the above example, suppose a solution of the *Embedding-Edge LP* is $x(B_1) = 1$ and $x(B_2) = 0.5$. Then $N = 2$, and $n(B_1) = 2, n(B_2) = 1$. Each data stream is divided into frames of 3 symbols, out of which the first 2 symbols flow over B_1 and the last symbol flows over B_2. In the k-th frame, the link uw carries only one non-empty subframe for B_2 containing one 'X' symbol. That subframe $\mathbf{y}_{uw,B_2,X}^{k-1}$ corresponds to the last symbol of the $(k-1)$-th frame of data. The link wt carries one subframe of two '$X+Y$' symbols for B_1 and another subframe of one '$X+Y$' symbol for B_2. These subframes $\mathbf{y}_{wt,B_1,X+Y}^{k-2}$,$\mathbf{y}_{wt,B_2,X+Y}^{k-2}$ correspond to the first two symbols of the $(k-2)$-th data frame and the last symbol of the $(k-2)$-th data frame respectively.

To implement the protocol, any node u needs to know N, $n(B)$ for all embeddings with non-zero $n(B)$, and $d(uv, B, \theta)$ and $d(vu, B, \theta)$ for all such embeddings B, $\theta \in \Gamma, v \in \aleph(u)$. The values of $d(uv, B, \theta)$ can be found in $O(nb|\Gamma|)$ time, where b is the number of embeddings for which $n(B) > 0$. In the following, we give the sequence of actions taken by any node u.

1. The node maintains an input queue for each (B, θ) pair for which $d(vu, B, \theta) < \infty$ for some $v \in \aleph(u)$.

2. For the k-th frame received from v on the link vu, the node u knows the 'composition', i.e., how many symbols for which (B, θ) pair are received on that frame and in what order. This is because the frame contains a non-empty subframe corresponding to (B, θ) if and only if $d(vu, B, \theta) \leq k$. Such a non-empty frame contains exactly $n(B)$ symbols. The transmission of all the non-empty frames is ordered in the lexicographic ordering of (B, θ). For any received frame on any link, u puts each received subframe in its respective input queue. If u is a source, it also takes the K generated symbols and creates the subframes of lengths $n(B)$ for all the relevant embeddings. Those are also placed in respective queues.

3. After queueing all the received and generated data in the k-th frame, u prepares the data to be transmitted on each link uv in the next, that is $(k+1)$-th, frame of N transmissions. The non-empty subframes for this transmitted frame are those for which $d(uv, B, \theta) \leq k+1$. If there is an input queue for (B, θ), i.e., if such a data subframe is received at u, then this subframe of $n(B)$ symbols is taken from the respective input queue. Otherwise, this subframe is generated from the subframes from the queues for $(B, \eta); \eta \in \Phi_{\uparrow}(\theta)$. If such a queue for (B, η) contains multiple subframes of $n(B)$ symbols, then the oldest of them is taken. For instance, in our example (Fig. 7), for constructing the subframe $\mathbf{y}_{wt,B_2,X+Y}^{k}$ at w for the k-th frame, w takes a subframe from its input queue (B_2, X) and a subframe from the input queue (B_2, Y) and adds them. At this time, in the first queue, there is only one subframe $\mathbf{y}_{uw,B_2,X}^{k-2}$ which is used now. But in the second queue, there are two subframes $\mathbf{y}_{vw,B_2,Y}^{k-1}$ and $\mathbf{y}_{vw,B_2,Y}^{k-2}$ available, out of which the older subframe $\mathbf{y}_{vw,B_2,Y}^{k-2}$ is used.

REFERENCES

[1] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas, "Fast approximation algorithms for multicommodity flow problems," *Journal of Computer and System Sciences*, vol. 50, no. 2, pp. 228–243, April 1995.

[2] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," *Journal of the ACM*, vol. 37, no. 2, pp. 318–334, April 1990.

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

[4] N. Garg and J. Konemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *IEEE Symposium Foundations of Computer Science*, 1998.

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

[6] E. Kushilevitz and Y. Mansour, "Computation in noisy radio networks," in *Proc. of SODA*, 1998, pp. 236–243.

[7] U. Feige and J. Kilian, "Finding OR in noisy broadcast network," *Information Processing Letters*, vol. 73, no. 1–2, pp. 69–75, January 2000.

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

[9] L. Ying, R. Srikant, and G. Dullerud, "Distributed symmetric function computation in noisy wireless sensor networks with binary data," in *Proc. of WiOpt*, April 2006.

[10] Y. Kanoria and D. Manjunath, "On distributed computation in noisy random planar networks," in *Proc. of IEEE ISIT*, Nice, France, June 2007.

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

[12] J. Korner and K. Marton, "How to encode the modulo-two sum of binary sources," *IEEE Trans. Inform. Theory*, vol. 25, no. 2, pp. 219–221, 1979.

[13] T. S. Han and K. Kobayashi, "A dichotomy of functions $f(x, y)$ of correlated sources (x, y)," *IEEE Trans. Inform. Theory*, vol. 33, no. 1, pp. 69–86, 1987.

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

[15] H. Feng, M. Effros, and S. A. Savari, "Functional source coding for networks with receiver side information," in *Proceedings of the Allerton Conference on Communication, Control, and Computing*, September 2004.

[16] Soheil Feizi and Muriel Medard, "On network functional compression," *available at http://arxiv.org/abs/1011.5496*, 2011.

[17] Brijesh Kumar Rai and Bikash Kumar Dey, "On network coding for sum-networks," *IEEE Trans. Inform. Theory*, vol. 58, no. 1, pp. 50–63, 2012.

[18] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing: Cut-set bounds," *IEEE Trans. Inform. Theory*, vol. 50, no. 2, pp. 1015–1030, 2011.

[19] Michael Langberg and A. Ramamoorthy, "Communicating the sum of sources in a 3-sources/3-terminals network," in *Proc. ISIT*, Seoul, Korea, 2009.

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

[21] O. Wohlmuth and F. Mayer-Lindenberg, "A method for them embedding of arbitrary communication topologies into configurable parallel computers," in *Proceedings of the 1998 ACM Symposium on Applied Computing*, 1998, pp. 569–574.

[22] V. Heun and E. W. Mayr, "Efficient dynamic embeddings of arbitrary binary trees into hypercubes," *Journal of Algorithms*, vol. 43, pp. 51–84, 2002.

[23] S. Kannan and P. Viswanath, "Multi-session Function Computation in Undirected Graphs," available at *http://www.ifp.illinois.edu/∼pramodv/pubs.html*

[24] D. Manjunath, Anurag Kumar and Joy Kuri, *Communication Networking: An Analytical Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[25] M. Saad, T. Terlaky, A. Vannelli, and H. Zhang, "Packing trees in communication networks," *J. Comb. Optim.*, vol. 16, pp. 402–423, 2008.

Virag Shah is currently working towards his PhD at The University of Texas at Austin. He received his Bachelor of Engineering degree in Electronics from University of Mumbai, India in 2007. He received his Master of Engineering degree in Telecommunications from Indian Institute of Science, Bangalore, India in 2009. From November 2009 to July 2010, he was Research Fellow at Indian Institute of Technology, Bombay, India working in the area of telecommunication networks. His research interests include design and analysis of telecommunication networks, queuing theory, network control, graph theory and information theory.

Bikash Kumar Dey (S'00-M'04) received his B.E. degree in Electronics and Telecommunication Engineering from Bengal Engineering College, Howrah, India, in 1996. He received his M.E. degree in Signal Processing and Ph.D. in Electrical Communication Engineering from the Indian Institute of Science in 1999 and 2003 respectively.

From August 1996 to June 1997, he worked at Wipro Infotech Global R&D. In February 2003, he joined Hellosoft India Pvt. Ltd. as a Technical Member. In June 2003, he joined the International Institute of Information Technology, Hyderabad, India, as Assistant Professor. In May 2005, he joined the Department of Electrical Engineering of Indian Institute of Technology Bombay where he works as Associate Professor. His research interests include information theory, coding theory, and wireless communication.

He was awarded the Prof. I.S.N. Murthy Medal from IISc as the best M.E. student in the Department of Electrical Communication Engineering and Electrical Engineering for 1998-1999 and Alumni Medal for the best Ph.D. thesis in the Division of Electrical Sciences for 2003-2004. He is a convenor of the Bharti Centre for Communication at IIT Bombay.

D. Manjunath received his BE from Mysore University, MS from IIT Madras and PhD from Rensselaer Polytechnic Institute in 1986, 1989 and 1993 respectively. He has been with the Electrical Engineering Department of IIT Bombay since July 1998 where he is now a Professor. He has previously worked in the Corporate R & D Center of GE in Schenectady NY (1990), Computer and Information Sciences Department of the University of Delaware (1992–93), Computer Science Department, University of Toronto (1993–94) and the Department of Electrical Engineering of IIT Kanpur (1994–98). His research interests are in the general areas of communication networks and performance analysis. His recent research has concentrated on in-network computation and random networks with applications in wireless and sensor networks, network pricing and queue control. He is a coauthor of two textbooks, *Communication Networking: An Analytical Approach* (May 2004) and *Wireless Networking* (Apr 2008), both of which are published by Morgan-Kaufman Publishers. He also heads the Computer Centre and is the convenor of the Bharti Centre for Communication both at IIT Bombay.