Multi-session Function Computation and Multicasting in Undirected Graphs

Sreeram Kannan and Pramod Viswanath
Coordinated Science Laboratory and Dept. of ECE
University of Illinois, Urbana-Champaign, IL 618l01
Email: {kannan1, pramodv}@illinois.edu

***Abstract*—In the function computation problem, certain nodes of an undirected graph have access to independent data, while some other nodes of the graph require certain functions of the data; this model, motivated by sensor networks and cloud computing, is the focus of this paper. We study the maximum rates at which function computation is possible on a capacitated graph; the capacities on the edges of the graph impose constraints on the communication rate.**

We consider a simple class of computation strategies based on Steiner-tree packing (so-called *computation trees*), which does not involve block coding and has minimal delay. With a single terminal requiring function computation, computation trees are known to be optimal when the underlying graph is itself a directed tree, but have arbitrarily poor performance in general directed graphs. Our main result is that computation trees are *near optimal* for a wide class of function computation requirements even at *multiple* terminals in *undirected* graphs.

The key technical contribution involves connecting approximation algorithms for Steiner cuts in undirected graphs to the function computation problem. Furthermore, we show that existing algorithms for Steiner tree packings allow us to compute approximately optimal packings of computation trees in polynomial time. We also show a close connection between the function computation problem and a communication problem involving multiple multicasts.

I. Introduction

In several communication scenarios, a receiver is interested in computing a function of data from different agents spread over a network. For example, in a sensor network, a fusion node is interested in computing a function of the various sensors. Similarly, in a cloud computing scenario, the data may be stored in a distributed manner, with different types of information about the same record stored in different locations. In such a scenario, one may be interested in computing a function of the data.

The problem of function computation has been studied in the setting when there are many nodes that have independent data but only one destination demanding a certain function of the data. In this paper, we generalize this in two different directions. First, we consider the problem of multi-session function computation, where there are multiple independent function computation sessions, all sharing a common communication infrastructure. Second, we consider the problem of function multicasting, where many destinations want to compute the same function of the independent sources. In fact, we study the general version of the problem, which we call multi-session function multicasting, which comprises of many independent function multicasting sessions sharing a common communication infrastructure. We will use K to denote the number of sessions and S, D to be the number of source and destination nodes involved in each multicasting session.

We assume that the communication infrastructure is specified by an undirected capacitated graph. By an undirected graph, we mean that on each edge $e = uv$ with capacity $c(e)$, node u can communicate to node v and vice versa, such that the sum of the rates of communication in the two directions does not exceed $c(e)$.

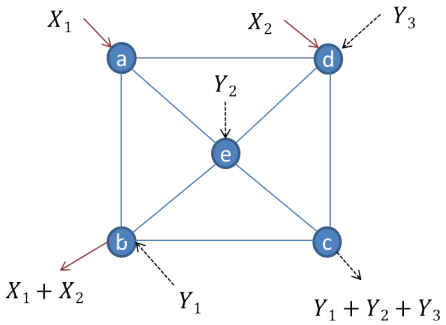


Fig. 1: Multi-session Example

We begin by considering the setting of multi-session function computation and then extend it to multi-session function multicasting. This scenario has several practical applications: in a sensor network, a communication infrastructure may be shared by several sensors of different modalities like heat, pressure, infrared and each of these modalities has a distinct or common fusion center.

For an instance of the multi-session function computation problem, see Fig. 1, where there are 2 sessions. In the first session, sensing nodes a and d have temperature information X_1 and X_2 and node b, which is the fusion node for temperature sensing, wants to compute the average temperature $\frac{X_1+X_2}{2}$. In the second session, sensing nodes b, e, d have pressure information Y_1, Y_2, Y_3 and node c, which is the fusion node for pressure sensors, wants to compute the average pressure $\frac{Y_1+Y_2+Y_3}{3}$.

We consider the setting of block function computation, where receiver i is interested in computing $R_i T$ function evaluations at the end of T time instants (the nodes in the network can forward arbitrary functions of data that they receive). The performance of the scheme is measured by the

set of rates $(R_1, ..., R_K)$, which can be achieved, this is called the computation capacity region $\mathcal{C}$.

A natural achievable strategy in this context is the use of *computation trees*. By computation tree, we mean strategies where a tree spanning the vertices involved in function computation is constructed (called a Steiner tree), and along the tree, in-network computations are performed in such a way that the function can be computed at the destination. Note that this is a very simple in-network computation strategy and does not involve block coding of data. Furthermore, this strategy avoids inter-session network coding, i.e., mixing of information across the different sessions (or modalities). These features make it a feasible practical scheme. We will show the surprising result that such a simple strategy can achieve near optimal performance.

The proposed achievable strategy is *universal*, i.e., it works for the computation of an arbitrary function. However, the performance of the strategy depends on a property of the function called λ-divisibility. A function f is said to be λ_f-*divisible* if the function can be computed in a divide-and-conquer fashion with every intermediate computation requiring at most λ_f symbols to store and communicate. In other words, every Steiner tree can be used for computation of the function, i.e., every Steiner tree is a computation tree with computation rate $\frac{1}{\lambda_f}$. Every function on S variables is λ_f-divisible for some $\lambda_f \leq \lceil S| $. For efficient computation of the function, we would like λ_f to be as small as possible.

It turns out that we can write a cut-set upper bound for computation of a class of functions that satisfy the marginal-injectivity property. A function is said to be *marginally injective*, if it is injective with respective to every variable, given any realization of the remaining variables. Note that, for a marginally injective function f, $\lambda_f \geq 1$. For example, linear functions over fields or groups are marginally injective, and have $\lambda_f = 1$.

In this paper, we show that for function computation in a graph $\mathcal{G}$ with K sessions and S non-overlapping sources per session, computation trees achieve $\mathcal{R}_{\text{comp}}$ such that

$$\frac{\bar{C}}{\lambda_f(S) g(S,K)} \subseteq \mathcal{R}_{\text{comp}} \subseteq \bar{C}, \tag{1}$$

where $\bar{C}$ is the cut-set bound, where

$$g(S,K) \leq \begin{cases} 1 & \text{if } \mathcal{G} \text{ is a tree} \\ 2 & \text{if } K = 1, \\ \kappa \log SK & \text{if } K > 1, \end{cases}$$

where κ is a universal constant. Furthermore, there are polynomial-time algorithms to find the computation trees that achieve these performance guarantees.

For the more general case of function multicasting, we get the following result. For K-session function multicasting with S sources and D destinations per session, computation trees achieve $\mathcal{R}_{\text{comp}}$ such that

$$\frac{\bar{C}}{2\lambda_f(S) g(S+D,K)} \subseteq \mathcal{R}_{\text{comp}} \subseteq \bar{C}, \tag{2}$$

where $g(S,K)$ is defined above. Note that the result for the general function multicasting case is only a factor of 2 weaker than the result for function computation.

These results are based on existing results for approximating "sparsest Steiner cuts" [2], [3], [4], and incorporates several special cases: when $S = 1$, it is the multiple unicast problem with K source-destination pairs, for which the seminal work of Leighton and Rao [12] shows a $O(\log K)$ gap between routing and cuts; when $K = 1$, it is the function computation scenario with a single receiver being interested in a function of S nodes. For all of these cases, we demonstrate that there is a close connection between the function computation problem and the multiple multicast communication problem, where there are K independent sources each of which wants to send common messages to a set of S destinations.

This result demonstrates that undirected graphs are fundamentally different from directed graphs in various aspects. While directed graphs with cycles are more general than undirected graphs, the special structure imposed by directed graphs allows very efficient packings of Steiner trees, thereby achieving performance close to the cut. For general directed graphs, Steiner tree packing can achieve a performance arbitrarily far away from the cut [32] when $K = 1$. For general K and $S = 1$, the problem reduces to that of multiple unicast for which Steiner tree packing reduces to routing. In this case, it is known that routing and the cut-set bound can be far away [24].

Furthermore, there are no known algorithms to compute the best Steiner tree packings within a constant factor approximation in directed graphs [31], whereas there is a polynomial time algorithm to compute a factor-2 approximation for Steiner tree packings in undirected graphs [7]. It shouldtip be noted that the presence of cycles in undirected graphs allows for interactive function computation. Nevertheless, our results show that, for a wide class of functions, simple non-interactive function computation is near optimal.

A. Related Work

1) Communication Complexity: The problem of function computation has been well studied in the communication complexity literature. The basic setup is that there are two nodes, having values x and y, and both of them wish to compute a function $f(x,y)$ of these nodes. The goal is to minimize the total number of bits communicated in the worst case. This problem was originally formulated by [19] and the reader is referred to [20] for a treatment of this problem. However, this setting does not allow for a block computation of the function.

2) Function Computation in Random Graphs: The problem of block function computation was originally formulated by Giridhar and Kumar [11]. They studied the scaling laws for computing several classes of functions in a random wireless network. In their work, the wireless nature of the medium is dealt with using scheduling (to avoid interference) and then in-network computation is performed by aggregating data through spanning trees. In contrast, in this paper, we do not deal with the wireless aspects and consider the problem of function computation on any specified undirected graph.

3) Computation Trees: Computation trees, where Steiner trees are used for computation, have been a popular strategy,

see [11], [10], [9], although the name was coined later in [39]. Such trees are optimal for function computation when the graph is itself just a directed tree [10], [9]. In an arbitrary directed acyclic graph, [9] proposed a strategy where several such trees can be packed and the rate of computation can be linked to the Steiner packing number. However, as pointed out earlier, in directed graphs, the Steiner packing number can be arbitrarily far away from the cut [32].

In undirected graphs, packing of specific computation trees resembling the structure of the function, were considered and algorithms for packing computation trees were provided in [39]. However, there are no guarantees on the computation rates achieved by this algorithm and it is unknown what the gap to capacity is.

4) Linear Coding: Another popular achievable strategy is linear coding where each intermediate node forwards a linear combination of the incoming symbols. For general directed graphs, when the function to be computed is linear, random linear coding is known to be optimal [33]. In [17], it is shown that linear coding is insufficient when the function to be computed is nonlinear and the potential loss due to employing linear coding is quantified.

5) Undirected Graphs: The problem of determining function computation capacity in undirected graphs is considered harder in general, due to the presence of cycles in the graph, which allow for *interaction* in function computation. Single-shot function computation in the 2-node setting has been a central problem of study in the field of communication complexity [19], [20]. Even allowing for block computation, for general functions, seemingly simple 2-node problems can become hard (see [40]). For a class of sum-threshold functions on undirected trees, this capacity is characterized in [10] using carefully orchestrated interactive strategies. In this paper, we show that for many function classes of practical interest, non-interactive function computation using computation trees can give near-optimal performance.

6) Function Multicasting: One particular problem that has attracted attention is the case where there are several sources, and each destination demands the sum of the sources [34], [35], [37]. In our terminology, we call this the function multicasting problem; in this case, the function happens to be linear as well.In [37], the case of a directed acyclic communication graph is considered and several negative results demonstrating the insufficiency of scalar and vector linear coding is shown, deriving inspiration from similar results for multiple unicast in directed networks. In contrast, in this paper, it is shown that if the communication graph is undirected, simple coding schemes can achieve within a constant factor of the optimal. Function multicasting was also considered in the different context of linear deterministic networks in [18].

7) Multi-session Function Computation: The case of function computation with multiple sessions has not received much attention, partly because even the single terminal scenario is sufficiently complicated and unsolved in general directed graphs. We are not aware of any existing work that studies this setting.

8) Correlated sources: Information theoretic approaches to function computation in simple settings with correlated sources was studied by [21], [22], [25]. Function computation through noisy channels was studied in [29], [30], [26] and through Gaussian wireless channels was studied in [27], [28]. However, these results are restricted to simple functions. In this paper, we only consider independent sources and noiseless channels but focus on general graphs and a wide class of functions.

B. Organization

We will first discuss the mathematical formulation of the multi-session function computation problem and describe the classes of functions considered in the paper with examples in Section II. Next, we describe our proposed scheme and outer bound in Section III. We state our main result and discuss its ramifications, especially the connection between the computation problem and a dual communication problem, in Section IV. We show how these results extend to the setting of multi-session function multicasting in Section V. We prove the main result in Section VI and conclude the paper in Section VII.

II. Problem Formulation

The communication network is represented by an undirected graph $\mathcal{G} = (V, E)$, and edge capacity function $c : E \to \mathbb{R}^+$. There are K "sessions", each session involving independent variables and the computation of a specified function at a terminal. The sources for session k are denoted by Σ_k, where

$$\Sigma_k = \{\sigma_{1k}, \sigma_{2k}..., \sigma_{Sk}\} \subseteq V, \quad k = 1, 2, ..., K, \tag{3}$$

each with its corresponding destination $\rho_k, k = 1, 2, .., K$. Define $G_k := \Sigma_k \cup \{\rho_k\}$ as the group of vertices involved in session k. We assume for notational simplicity that each session involves the same number of variables; in the general case that session k involves $|\Sigma_k|$ variables, we can replace SK in the results by $\sum_k |\Sigma_k|$ and the results will continue to hold.

The source σ_{ik} has several instances of an information variable X_{ik}, which takes values over a common alphabet $\mathcal{A}$. We do not impose a statistical structure on the messages. Alternatively, it is possible to set up the sources as independent random variables, in which case, the results in the paper will continue to hold with modification of notation. The destination ρ_k is interested in computing a function $f : \mathcal{A}^S \to \mathcal{B}$, i.e., it wishes to reconstruct $f(X_{1k}, X_{2k}.., X_{Sk})$. While Σ_k and $\Sigma_{k'}$ could in general overlap, we assume that the information of interest to destination ρ_k is distinct from the information of interest to destination $\rho_{k'}$.

This function computation happens several times and therefore we consider a block function computation problem as follows. The network can potentially employ in-network computation (network coding) in order to compute the function. Formally, a coding scheme of block length T and achieving a rate tuple $(R_1, ..., R_k)$ is described as follows:

- The source σ_{ik} has messages $X_{ik}(1), X_{ik}(2), ..., X_{ik}(R_kT)$. The destination ρ_k is interested in computing $f(X_{1k}(t), X_{2k}(t), ..., X_{Sk}(t))$ for $t = 1, 2, ..., R_kT$. The message set of source $v = \sigma_{ik}$

TABLE I: Function Examples

	Function Name	$f(x_1, ..., x_S)$	$\lambda_f(S)$-divisible	Marginally Injective?
1	Linear (field)	$x_1 \oplus x_2 \oplus ...x_S$	1	✓
2	Addition (abelian group)	$x_1 \oplus x_2 \oplus ...x_S$	1	✓
3	Arithmetic Sum ($A := \|\mathcal{A}\|$)	$x_1 + ... + x_S$	$\log_A\{S(A-1)+1\}$	✓
4	Real Sum over $\mathcal{A} = \{0 : \delta : 1\}$	$x_1 + ... + x_S$	$\log_A\{S(A-1)+1\}$	✓
sds 5	ℓ_p-norm suitably quantized	$x_1^p + ... + x_S^p$	$\log_A\{S(A-1)+1\}$	✓
6	Histogram	Hist$(x_1, ..., x_S)$	$\log_A\{\binom{S+A-1}{S}\}$	✓
7	Symmetric	f$(x_1, ..., x_S)$	$\leq \log_A\{\binom{S+A-1}{S}\}$	Depends
8	Maximum	$\max(x_1, ..., x_S)$	1	No
9	Sum threshold	$\mathbb{1}_{[x_1+x_2+...+x_S>m]}$	$\log_A\{S(A-1)+1\}$	No

is denoted by W_{vk} where $|W_{vk}| = |\mathcal{A}|^{R_k T}$. For simplicity of notation, if a given node $v \notin \Sigma_k$, then we set $W_{vk} = \emptyset$.

- Since the network graph is undirected the capacity $c(e)$ on edge e has to be shared between the forward and reverse direction, let the fraction on forward direction be $\alpha(e)$. Once the α is fixed, then the network becomes a directed network. Let us scale this network to have integral capacities and represent this network as a *directed multigraph*, potentially having multiple edges between the same nodes.
- At each node v, at each time t, there is a mapping

$$g_{v,t} : \mathcal{A}^{|\text{In}(v)|(t-1)} \times W_{v1} \times W_{vk} \times ...W_{vK} \to \mathcal{A}^{|\text{Out}(v)|}, \quad (4)$$

where In(v) and Out(v) denote the set of incoming edges and the set of outgoing edges of node v. The function $g_{v,t}$ thus specifies a mapping from the messages on the incoming edges till time $t-1$ and the node's own messages to the outgoing edge message at time t.
- The decoding map at destination ρ_k is given as a function of its incoming edges till time t

$$\psi : \mathcal{A}^{|\text{In}(v)|T} \to \mathcal{B}^{R_k T}. \quad (5)$$

- The coding scheme is said to achieve rate tuple $(R_1, ..., R_K)$ if each destination correctly recovers the function of its desired nodes.

The set of all achievable rate tuples $(R_1, ..., R_K)$ is called the computation capacity region $\mathcal{C}$.

A. Function Classes

We will define certain classes of functions which we will be interested in. A function $f : \mathcal{A}^S \to \mathcal{B}$ is called λ_f-divisible, if for every index set $I \subseteq [S]$, there exists a finite set $\mathcal{B}_I$ and a function $f^I : \mathcal{A}^{|I|} \to \mathcal{B}_I$ such that the following hold:

1) $f^{[S]} = f$
2) $|f^I(.)| \leq |\mathcal{A}|^{\lambda_f}$.
3) For every partition $\{I_1, ..., I_j\}$ of I, there exists a function $g : \mathcal{B}_{I_1} \times \cdots \times \mathcal{B}_{I_j} \to \mathcal{B}_I$ such that for every $x \in \mathcal{A}^{|}I|$,

$$f^I(x) = g(f^{I_1}(x_{I_1}), ..., f^{I_j}(x_{I_j})). \quad (6)$$

If a function is λ_f-divisible, then it means that it can be computed in a divide-and-conquer manner such that every intermediate computation needs only λ_f symbols to store and transmit. Every function is $|S|$-divisible in a trivial manner, since retaining all the information is sufficient to compute the function. Our interest will be in functions f for which λ_f is small. λ_f-divisible functions can be seen to be equivalent to λ_f-bounded functions defined in [9], we prefer the alternate name and definition since it is more suggestive. Divisible functions as defined in [11], [38] and [9] are λ_f-divisible with $\lambda_f = \log_{|\mathcal{A}|} |\mathcal{B}|$. For example, a linear function over a finite field has $\lambda_f = 1$. We refer the reader to Table I for a listing of λ_f for various functions.

A function $f : \mathcal{A}^S \to \mathcal{B}$ is called *marginally injective* if, for any variable i, for any fixed assignment on the other variables, the function can take on $\mathcal{A}$ distinct values as x_i varies over $\mathcal{A}$, i.e.,

$$\forall i, \forall y_{[S]\setminus i} \in \mathcal{A}^{S-1}, \psi : \mathcal{A} \to \mathcal{B}, \quad (7)$$

defined by $\psi(x_i) = f(x_i, y_{[S]\setminus i})$, is injective.

Several functions of interest satisfy this property, as listed in Table I. An example of a function which is *not* marginally injective is the max-function over an ordered set. The value of the maximum does not depend on any other variable if one of the variable is assigned the maximum possible value.

B. Examples

Various examples of functions are provided in Table I. Also listed is whether the function is marginally injective or not, and the value of λ_f for which f is λ_f-divisible (the value of λ_f could in general depend on the number of variables S).

1) The linear function over a finite field is an obvious example of a function which is 1-divisible and marginally injective. For linear function computation with a single terminal, linear coding is known to be optimal [33], but the case of multiple terminals has not been studied.
2) The case of addition over an Abelian group is very similar to the finite field sum. However, for addition over an Abelian group, existing random linear coding techniques do not apply due to the lack of the field structure, whereas our computation tree based approach naturally extends to this case.
3) The arithmetic sum is $\log_A B$-divisible since maintaining the arithmetic sum of the subsets is sufficient, and it is also marginally injective.
4) Real sum over a quantized alphabet on [0,1] (quantized to a fixed precision δ) is only a disguised version of the arithmetic sum since after scaling by $\frac{1}{\delta}$, we have converted it into the arithmetic sum. If precise quantization is not required in the application, then we can maintain quantized versions of the sum (to accuracy δ) throughout the computation tree, thus having an effective λ of 1. This function is practically relevant in sensor networks, since sum of the LLR (log-likelihood-ratio) is a sufficient statistic in some cases [41].
5) ℓ_p-norm is basically just a real sum, except for the fact that we need x_i^p to be quantized to a precision of δ. This is again a practically important function in sensor networks.
6) The histogram is a function from $\mathcal{A}^S$ to $\mathcal{B}$ with $A := |\mathcal{A}|$ and $B := |\mathcal{B}|$. Using a simple enumeration it can be computed that the histogram can take on one of $B = \binom{S+A-1}{S}$ values. The histogram is $\log_A B$-divisible because, for any subset, the histogram of the subset is a sufficient statistic to maintain. For the case of computing a histogram of S nodes at a single terminal, our method leads to a $2\lambda_f = 2\log_A \binom{S+A-1}{s}$ approximation as opposed to the method in [9] of using the binary arithmetic sum, which leads to a bigger factor gap of $(A-1)\log_A(AS)$ (adapted to undirected graphs). For example, when $A = 16$, $S = 50$, $2\lambda_f = 23.6$, whereas $(A-1)\log_A(AS) = 36.1$.
7) Any symmetric function (which is invariant to permutations of input symbols) depends only on the histogram. Thus the histogram is a sufficient statistic to compute any symmetric function.
8) Maximum is an example of a function which is clearly 1-divisible, however it is not marginally injective. Consider the alphabet $0, 1, 2, .., a-1$. If we know that $\max(x_2, ..., x_S) = A-1$, then the function f no longer depends on x_1 and is therefore not marginally injective. While the achievable strategies in this paper continue to hold, the outer bound is no longer valid for this function.
9) Sum-threshold functions are basically of the form $\mathbb{1}_{[x_1+x_2+\ldots x_S>m]}$. The arithmetic sum of the variables is a sufficient statistic to compute the function and hence the function has
$$\lambda_f \leq \log_A\{S(A-1)+1\}. \tag{8}$$
The function is not marginally injective, because, if the variables other than x_i have a sum of greater than m, the function no longer depends on x_i.

III. Inner and Outer Bounds

In this section, we present our achievable scheme based on computation trees and the outer-bound on the rates for multi-session function computation in undirected graphs. Similar bounds have been previously studied in the context of single-session function computation in directed acyclic graphs [9].

A. Cut-set Bound

The cut-set bound defined in [9] is special to directed acyclic graphs and does not generalize to cyclic graphs. We use the technical condition of *marginal injectivity* in order to establish a simple cut bound on the communication rate based on the separation of one node from the rest of the nodes in the session. Note that we call this simple bound as the cut-set bound in this paper. This bound can be strengthened by taking into account the effect of separating multiple nodes from each other; however, we do not need that generality to prove the results of this paper.

Given a set $\Omega \subseteq V$, define
$$K(\Omega) := \{k : G_k \cap \Omega \neq \emptyset, G_k \cap \Omega^c \neq \emptyset\}, \tag{9}$$
as the set of sessions disconnected by Ω and define
$$\text{Cut}(\Omega) := \sum_{(ij)\in E: i\in\Omega, j\in\Omega^c} c(ij). \tag{10}$$
Then, for any scheme computing a *marginally injective* function, for any $\Omega \subseteq V$, define
$$\bar{\mathcal{C}} = \{\overline{R} : \sum_{k\in K(\Omega)} R_k \leq \text{Cut}(\Omega)\ \forall\Omega\}. \tag{11}$$

The main observation is that $\mathcal{C} \subseteq \bar{\mathcal{C}}$. We will prove this for the case of $K = 1$, the general case is similar. When $K = 1$, we only need to consider Ω such that $K(\Omega) = 1$, i.e., the cut separates G_k. Let Ω be such that $\rho_1 \in \Omega^c$ and for some i, $\sigma_{i1} \in \Omega$. The information on edges between Ω and Ω^c can take $\mathcal{A}^{\text{Cut}(\Omega)T}$ possible values. The function is marginally injective, and therefore this should atleast convey as much information as one of the sources, σ_{1i}. Thus
$$|\mathcal{A}|^{\text{Cut}(\Omega)T} \geq |\mathcal{A}|^{R_1 T}, \tag{12}$$
and so $R_1 \leq \text{Cut}(\Omega)$ which proves the required bound.

Informally, the marginal-injectivity property suggests that even in the presence of feedback or interaction, each node needs to convey its information symbol in order for function computation to succeed. Therefore, we can think of marginally-injective functions as functions for which interaction does not help much. On the other hand, if a function is not marginally injective, interaction can in general help.

B. Achievable Strategy

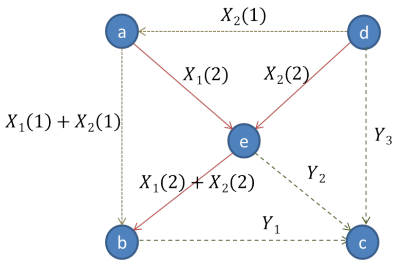


Fig. 2: Multi-session Example

We first formally define Steiner trees. An *undirected Steiner tree* on a set G is defined as an undirected tree τ which includes G in its vertex set. A *directed Steiner tree* rooted at ρ on a vertex set S is defined as a directed tree rooted at ρ that includes S in its vertex set. Given an undirected Steiner tree on $G_k := \{\rho_k\} \cup \Sigma_k$, there is a (unique) orientation of edges such that we get a directed Steiner tree on Σ_k rooted at ρ_k. Note that whenever we mention about Steiner tree, we refer to a tree with unit capacity edges.

Let $\mathcal{T}_k$ be the set of Steiner trees on G_k and $\mathcal{T} = \cup_k \mathcal{T}_k$. A fractional Steiner packing of $\pi_1, ..., \pi_K$ is said to be achievable if so many trees can be packed simultaneously: i.e., there exist $f_\tau, \tau \in \mathcal{T}$ such that

$$\pi_k = \sum_{\tau \in \mathcal{T}_k} f_\tau \quad k \in [K] \tag{13}$$

$$\sum_{\tau \in \mathcal{T}: e \in \tau} f_\tau \leq c(e) \quad \forall e \in E. \tag{14}$$

Let $\mathcal{R}_s$ be the Steiner packing rate region, i.e., the set of all fractional Steiner packing rates $(\pi_1, ..., \pi_K)$, which are achievable.

The achievable strategy is based on using computation trees. The key observation is that, if a function is λ_f-divisible, then any Steiner tree with unit capacity can be used to compute the function at a rate $\frac{1}{\lambda_f}$. Thus a rate tuple

$$(R_1, ..., R_K) = \frac{1}{\lambda_f}(\pi_1, \pi_2, ..., \pi_K), \tag{15}$$

can be achieved for computing any λ_f*-divisible function*. We denote the set of all rate tuples achievable by this computation tree scheme as $\mathcal{R}_{\text{comp}}$. Thus

$$\mathcal{R}_{\text{comp}} = \frac{\mathcal{R}_s}{\lambda_f}. \tag{16}$$

For a demonstration of our achievable strategy for the example function computation problem in Fig. 1, see Fig. 2. In this strategy, there are two Steiner trees for session 1, the first one is comprised of edges da and ab, and the second one is comprised of edges ae, de and eb. There is one Steiner tree for session 2 which has edges bc, ec and dc. This Steiner packing strategy achieves the best sum rate among all possible strategies. This can be observed by taking the cut separating b from the rest of the nodes; since the cut separates both sessions the corresponding value of the cut is a bound on the sum rate. Note that in general, in our achievable strategy, the capacity of each edge could be shared between several Steiner trees; this is called fractional Steiner packing, as opposed to the integral Steiner packing demonstrated in this example.

We note that both our achievable rate and outer bound depend only on the definition of the sets G_k and not on the particular way in which $G_k = \Sigma_k \cup \{\rho_k\}$ is divided into sources Σ_k and the destination ρ_k. Thus if the destination ρ_k swaps its role with one of the sources in Σ_k, the achievable rate and the outer bound remains the same. It is not clear if this property holds for any achievable strategy and outer bound; in particular, it is interesting to study this question for the capacity region.

IV. Main Result

We first state our main result, which shows a factor approximation for the capacity region.

Theorem 1. *For computation of λ-divisible functions in a graph $\mathcal{G}$ with SK sources and K terminals demanding functions of S non-overlapping variables, computation trees achieve $\mathcal{R}_{comp}$ such that*

$$\frac{\bar{\mathcal{C}}}{\lambda_f(S) g(S,K)} \subseteq \mathcal{R}_{comp}, \tag{17}$$

where $\bar{C}$ is the cut-set bound, and

$$g(S,K) \leq \begin{cases} 1 & \text{if } \mathcal{G} \text{ is a tree} \\ 2 & \text{if } K = 1, \\ \kappa \log SK & \text{if } K > 1, \end{cases}$$

where κ is a universal constant that does not depend on the number of nodes in the network or the specific function to be computed. Furthermore, if the function is marginally injective, $\mathcal{C} \subseteq \bar{\mathcal{C}}$.

In the most general case, this result shows that our achievable strategy is optimal to within a factor $\lambda_f(S) O(\log SK)$. For functions which have a small λ_f, (for example, linear functions which have $\lambda_f(S) = 1$), the dominating factor in the approximation is $O(\log SK)$. We will compare this to some simple bounds obtainable by other methods.

Firstly for $S = K = 1$, the max-flow min-cut theorem states that the rate suggested by cut-set bound is achievable. Now one simple strategy is to time-share between the various sessions; this will take K time instants. Furthermore, in each session, the function can be computed by first routing each source to the sink and then computing the function at the destination. This process will take a total of SK time instants to achieve the cut-set bound; thus the cut-set bound is achievable within a factor SK by using this strategy. A smarter strategy is to choose one source per session to communicate to its corresponding sink simultaneously, so that we get a multiple unicast problem, for which Leighton and Rao [12] showed a $O(\log K)$ gap between flows and cuts. Since we are time sharing between the S sources, we get a factor $O(S \log K)$ between our achievable strategy and the cut-set bound. In comparison to these results, we see that our approximation factor of $O(\log SK)$ is much stronger. Furthermore, we observe that a well provisioned network will have capacities scaling linearly with K, the number of sessions. Thus, the cut-set bound will scale as K

and hence our achievable rates will scale at least on the order of $\frac{K}{\log SK}$.

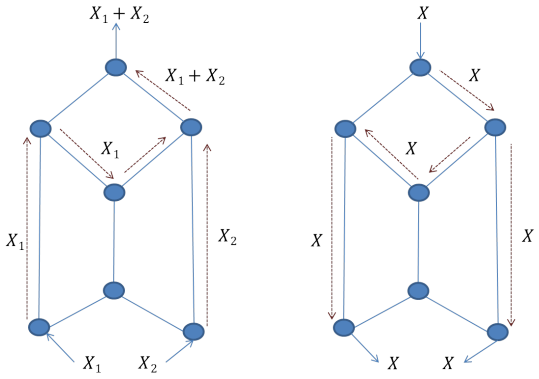
Theorem 1 is proved by first showing a connection between Steiner packing rates and the cut-set bound,

$$\frac{\bar{\mathcal{C}}}{g(S,K)} \subseteq \mathcal{R}_s \subseteq \bar{\mathcal{C}}, \tag{18}$$

where $\mathcal{R}_s$ is the Steiner packing rate region, and using the fact that

$$\mathcal{R}_{\text{comp}} = \frac{\mathcal{R}_s}{\lambda_f(S)}. \tag{19}$$

A. Relation between computation and communication



(a) A Network for Function Computation

(b) A Multicast Network

Fig. 3: Dual Communication and Computation Networks

Our results are strongly motivated by an analogy between the function computation problem and a multi-session communication problem, and analogous results available for the communication problem. This "dual" communication problem is obtained by reversing the nature of the sources and destinations. In particular, in a function computation session, a single sink node computes the function of many sources; in the communication problem, many destinations demand the same information from the single source (this is called multicasting). The key observation is that for the computation problem, every Steiner tree is a computation tree and for the communication problem, every Steiner tree is a multicasting tree. In the computation problem, whenever information streams merge in the Steiner tree, in-network computation is performed. In the dual communication problem, whenever information streams bifurcate in the Steiner tree, replication is performed. Thus, viewed in this manner, *replication and computation are dual operations*. Therefore, the achievable rates have a natural relationship. It turns out that the cut-set bound for the computation problem in this paper also has a natural analog for the communication problem.

Formally, given a function computation network, we will define the communication problem as the following: the communication graph G is the same with the same capacities. In the computation problem, there are K functions to be computed. In the communication problem, there are K independent messages to be communicated. For each message k, there is a source ρ^k and S destinations defined by the set Σ_k, each of which desire the message. The k messages are independent. This problem is called as the *multiple multicasting problem* in the network coding literature. Thus, for every multi-session computation problem, there is a dual multiple multicast communication problem.

We refer the reader to Fig. 3 for an example of a function computation problem over an undirected butterfly network, and the dual problem of multicasting. In Fig. 3a, there are two sources with messages X_1 and X_2 and one destination demanding $X_1 + X_2$. A Steiner tree connecting the three nodes is also shown in the figure with dotted lines. Till the two streams meet, the information X_1 and X_2 are passed along separately, and when they meet the function $X_1 + X_2$ is computed and forwarded along. The dual communication problem in Fig. 3b has a single source X which needs to be multicast to two destinations. Again, the Steiner tree is shown in dotted lines; in this multicast problem, when the Steiner tree bifurcates, the information X is replicated on both the outgoing edges.

1) Achievable Strategy: We will now see that both our achievable strategy and the cut-set bound for the computation problem have a natural analog in the communication problem as well. First, we focus on the achievable strategy. For the computation problem, our achievable strategy is based on packing Steiner trees for the various G_k. For the communication problem also, packing Steiner trees forms a natural achievable strategy, since a Steiner tree can be used to disseminate a message from one of the nodes in G_k, namely ρ_k to the rest of the nodes Σ_k. Thus given any fractional Steiner packing $(\pi_1, ..., \pi_K)$, a rate tuple $(\pi_1, ..., \pi_K)$ can be achieved for the multiple-multicasting problem. Let $\mathcal{R}_{\text{m.m.}}$ denote the achievable region for the multiple multicast problem. By the above observation, this equals the Steiner packing region, i.e., $\mathcal{R}_{\text{m.m.}} = \mathcal{R}_s$.

2) Outer Bound: Given any set $\Omega \subseteq V$, define $K(\Omega)$ as before $K(\Omega) := \{k : G_k \cap \Omega \neq \emptyset, G_k \cap \Omega^c \neq \emptyset\}$ as the set of sessions disconnected by Ω and define $\text{Cut}(\Omega) := \sum_{(ij)\in E: i\in\Omega, j\in\Omega^c} c(ij)$. This implies that if we separate Ω and Ω^c, for each $k \in K(\Omega)$, at least one destination is separated from the source and thus

$$\bar{\mathcal{C}} = \{\overline{R} : \sum_{k\in K(\Omega)} R_k \leq \text{Cut}(\Omega)\ \forall\Omega\}, \tag{20}$$

is an outer bound on the set of rates achievable for the multiple-multicast problem.

3) Capacity Approximation: Since the achievable strategy is given by Steiner packing and the outer bound is given by cut-set bound, we can now bound the gap between the two as,

$$\frac{\bar{\mathcal{C}}}{g(S,K)} \leq \mathcal{R}_{\text{m.m.}} \leq \bar{\mathcal{C}}, \tag{21}$$

where $g(S,K)$ is as defined in Theorem 1. Thus we see that the function computation problem and the communication problem have a natural duality for the proposed achievable strategy and the outer bound.

4) Linear Function Computation: Consider the special case when the source alphabet $\mathcal{A}$ is a field and the function to be computed is linear over the field. The linear function has $\lambda_f = 1$ and is marginally injective (see Sec. II-B). Thus we have

$$\frac{\bar{\mathcal{C}}}{g(S,K)} \subseteq \mathcal{R}_{\text{comp}} \subseteq \bar{\mathcal{C}}. \tag{22}$$

For the case of a single session $K = 1$, $g(S, K) \leq 2$, which implies that the achievable rate region using Steiner packings and the cut-set bound are approximately the same, to within a factor of 2. This is the dual of the corresponding result for multicasting [13], which shows that Steiner packing and cut are within a factor of 2, thus bounding the gain due to network coding in undirected graphs.

The single-session linear-function-computation problem has been well studied, and for directed graphs (potentially cyclic), it is known that the cut-set bound is achievable [33]. The achievable strategy is given by random linear coding and is inspired by the strategy used for the dual multicasting problem. The duality between linear coding for single-session linear-function computation and linear coding for multicasting was observed in [33]. However, there is no natural way to extend this duality (or even the achievable strategy) to the computation of general functions. Furthermore, our strategy of Steiner packings enables us to tackle the case of multi-session linear-function computation and show a provable approximation ratio of $\kappa(\log SK)$ between the achievable strategy and the cut-set bound.

B. Proof of Theorem 1 for the special case, $K = 1$

In this setup, one node wants to compute a function of all the sources. Since $K = 1$, the regions collapse to single numbers, for which we use small case letters. There is a close relationship between the fractional Steiner packing number $R_s = \pi$ and the cut $\bar{C}$ (which is also called the Steiner cut). This relationship is based on the Tutte-Nash-Williams theorem [14], [15] and Mader's undirected splitting-off theorem [16] and was elucidated in the multicast setting by Li and Li [13]:

$$\frac{1}{2}\bar{C} \leq R_s \leq \bar{C}. \tag{23}$$

Therefore, by using this fractional Steiner packing, and $R_{\text{comp}} = \frac{1}{\lambda_f} R_s$, we get the desired result,

$$\frac{1}{2\lambda_f}\bar{C} \subseteq R_{\text{comp}}. \tag{24}$$

V. Multi-Session Function Multicasting

A natural duality relationship between function computation and multicasting was shown in Sec. IV-A. In this section we extend our theory to encompass these distinct problems into the general problem of function multicasting.

In the general problem of multi-session function multicasting, there are K sessions. For session k, there is a set of S sources $\Sigma_k = \{\sigma_{1k}, ..., \sigma_{Sk}\}$ and a set of D destinations $P_k = \{\rho_{1k}, ..., \rho_{Dk}\}$. Each source σ_{ik} has several instances of an information variable X_{ik} over a common alphabet $\mathcal{A}$. For session k, each destination $\rho_{ik}, \quad \forall i$ wants to compute the *same* function of the sources $f(X_{1k}, ..., X_{Sk})$ taking values in alphabet $\mathcal{B}$.

This problem captures the multi-session function computation problem as a special case by setting $D = 1$, i.e., there is only one destination per session and it captures the multi-session multicasting (also called multiple multicasting) as a special case by setting $S = 1$, i.e., there is only one source per session and all the destinations demand the identity function of that source.

A. Outer Bound

Define $G_k = \Sigma_k \cup P_k$ to be the group of terminals associated with session k (note that this definition of G_k generalizes the definition in Sec. II). Consider any cut that separates one node in G_k from the rest. If this node happens to be a destination, the rate of function multicasting is upper bounded by the value of the cut since information needs to be delivered to this destination. Alternately if this node happens to be a source, then the rate of function computation is upper bounded by the value of the cut under the marginal injectivity assumption. Thus the cut-set bound proposed in Sec. III-A is valid for the function multicasting problem under the same marginal injectivity assumption and the updated definition of G_k. Thus $\mathcal{C} \subseteq \bar{\mathcal{C}}$, where $\mathcal{C}$ is the capacity region of the multi-session function multicasting problem and $\bar{\mathcal{C}}$ is as defined in Sec. II.

B. Achievable Strategy

As in the case of multi-session function computation, the achievable strategy keeps information flow separate for each session, i.e., there is no inter-session network coding. The key idea of the proposed strategy is to break down the problem of function multicasting into two distinct tasks:

- First, the requisite function is computed at one of the destination nodes. This is done using a computation tree constructed from a Steiner packing.
- Second, the computed function is multicast to all the other destination nodes.

For doing the first task, we need a Steiner tree connecting the sources to a specific destination. For accomplishing the second task, we need a Steiner tree connecting all the destinations. In order to find the best achievable rate among all such strategies, we may have to further optimize the particular node at which the function is initially computed. This optimization problem may turn out to be hard to solve and therefore we resort to a further simplification.

In our proposed scheme, the destination at which the function is computed in the first stage is not optimized but chosen arbitrarily from P_k, let us fix it to ρ_{1k}, the first element of the set P_k. We first construct a Steiner tree between all the sources and all the destinations, i.e., a Steiner tree spanning the set G_k of nodes. We then use this Steiner tree in two phases to do function multicasting. In the first phase, the Steiner tree is used to compute the function at ρ_{1k} (by using the achievable strategy described in Sec. III-B for function computation). By the analysis in Sec. III-B, if we have a

Steiner tree of rate $\lambda_f(S)$, we can do function computation at rate 1. Next, we use the same Steiner tree to do multicasting of the function. The function takes value in the alphabet $\mathcal{B}$ while the source symbols are from an alphabet $\mathcal{A}$. Thus, if we are given a Steiner tree of rate $\log_{|\mathcal{A}|}|\mathcal{B}|$, we can multicast the computed function at rate 1. Thus, in order to do function multicasting at rate 1, we need a Steiner tree of rate $\lambda_f(S)+\log_{|\mathcal{A}|}|\mathcal{B}| \leq 2\lambda_f(S)$. Stated differently, given a Steiner tree of rate 1, we can do function multicasting at rate $\frac{1}{\lambda_f(S)+\log_{|\mathcal{A}|}|\mathcal{B}|} \geq \frac{1}{2\lambda_f(S)}$.

C. Performance of the Proposed Strategy

In this section, we examine the performance achieved by the proposed strategy. The rate region achievable for function multicasting using the proposed achievable strategy is related to the Steiner packing rate as follows,

$$R_{\text{mult}} = \frac{R_s}{\lambda_f(S)+\log_{|\mathcal{A}|}|\mathcal{B}|} \geq \frac{1}{2\lambda_f(S)}, \tag{25}$$

where R_s is the Steiner packing rate region. Now R_s is related to the cut-set bound as shown in (18),

$$\frac{\bar{\mathcal{C}}}{g(S+D,K)} \subseteq \mathcal{R}_s \subseteq \bar{\mathcal{C}}, \tag{26}$$

where

$$g(S+D,K) = \begin{cases} 1 & \text{if } \mathcal{G} \text{ is a tree} \\ 2 & \text{if } K=1, \\ \kappa(\log(S+D)K) & \text{if } K>1, \end{cases} \tag{27}$$

with κ a universal constant.

Thus we have proved the following theorem characterizing the capacity region of the function multicasting problem.

Theorem 2. *For the problem of function multicasting in a graph $\mathcal{G}$ with K sessions and each session having S sources and D destinations demanding marginally-injective functions of non-overlapping variables, the proposed strategy achieves rates $\mathcal{R}_{mult}$ such that*

$$\frac{\bar{\mathcal{C}}}{2\lambda_f(S)g(S+D,K)} \subseteq \mathcal{R}_{mult} \subseteq \mathcal{C} \subseteq \bar{\mathcal{C}}, \tag{28}$$

where $\mathcal{C}, \bar{\mathcal{C}}$ are the capacity region and the cut-set bound respectively and $g(S+D,K)$ is specified in (27)*.*

VI. Proof of Main Result

We will now prove this result for general K using connections to algorithmic work showing approximation algorithms for "sparsest Steiner cuts" [2], [3], [4].

We first give a description of rate regions using the max-concurrent flow representation in Sec. VI-A, which is fairly standard in the algorithmic literature. Then we compute the dual of this linear program in Sec. VI-B. This is used in Sec. VI-C to compute the gap between Steiner tree packing and cuts in tree networks. [1] In Sec. VI-D we state existing results that imply logarithmic gaps between Steiner packings and cuts. We also describe existing polynomial time algorithms that achieve these rate guarantees.

[1]While this calculation is elementary, to our knowledge, it is not available as it is elsewhere.

A. Description of Rate Regions

For the case of multiple sessions, we have to deal with rate regions and cut-set regions. In order to deal with these regions in a compact manner, we use the max-concurrent flow representation. In this representation, we deal with all possibly rays in the capacity region, we call the vector of length K, denoting a ray as a demand vector. For example, the ray given by the demand vector $(1,1,...,1)$ corresponds to the direction signifying symmetric rates for all the sessions. In general, let $(D_1,...,D_K)$ be a given demand vector for sessions 1,..,K. In this formulation, we want to achieve a rate proportional to the given demands, i.e., we want to achieve a rate tuple

$$(R_1,...,R_K) = \alpha(D_1,...,D_K), \tag{29}$$

where we would like to find the maximum value of α such that $\alpha(D_1,...,D_K)$ is in the capacity region $\mathcal{C}$. The idea here is that by understanding the best α for each demand vector is equivalent to characterizing the convex capacity region $\mathcal{C}$.

The achievable rate for the proposed scheme is given by $\frac{1}{\lambda_f}(\pi_1,...,\pi_K)$, where $(\pi_1,...,\pi_K)$ is a simultaneous fractional Steiner packing, i..e, we can pack π_i trees simultaneously for each session. We set $(\pi_1,...,\pi_K) = \gamma(D_1,...,D_K)$ and want to maximize γ, this problem is called the maximum concurrent Steiner flow problem. The maximum value γ^* is called the (maximum concurrent) Steiner packing rate.

The maximum concurrent Steiner flow problem can be written as

$$\gamma^* = \max \gamma \quad \text{s.t.} \tag{30}$$

$$\sum_{\tau \in \mathcal{T}_k} f_\tau \geq \gamma D_k \quad \forall k \in [K] \tag{31}$$

$$\sum_k \sum_{\tau \in \mathcal{T}_k : e \in \tau} f_\tau \leq c(e) \quad \forall e \in E. \tag{32}$$

The cut-set bound on $\mathcal{C}$ now translates to

$$\alpha \sum_{k \in K(\Omega)} D_k \leq \text{Cut}(\Omega) \tag{33}$$

$$\Rightarrow \alpha \leq \frac{\text{Cut}(\Omega)}{D(\Omega)} =: \nu(\Omega), \tag{34}$$

where $D(\Omega) := \sum_{k \in K(\Omega)} D_k$ is the total demand separated by the cut, and $\nu(\Omega)$ defined here is referred to as the sparsity of the cut Ω. Thus

$$\alpha \leq \nu^* := \min_{\Omega \subseteq V} \nu(\Omega), \tag{35}$$

where the minimizer Ω^* is called the *sparsest Steiner cut*. The cut-set bound also bounds the Steiner packing rate and hence, $\gamma^* \leq \nu^*$ (see [2]).

We note that while we have defined cuts using subset of the vertex set $\Omega \subseteq V$, there is an alternate way of defining the cuts using subsets of edges $F \subseteq E$. We define the sparsity of edge cut F as

$$\nu(F) = \frac{\text{Cut}(F)}{D(F)}, \tag{36}$$

where $\text{Cut}(F) = \sum_{f \in F} c(f)$ and $D(F) = \sum_{k \in K(F)} D_k$, and $K(F)$ is the set of sessions separated by F, i.e., at least one

node of G_k is disconnected from another node of G_k in the graph $(V, E\backslash F)$. In general, cuts based on edge sets and vertex sets can be very different, but for undirected graphs, the two turn out to be equivalent (see [6]), i.e.,

$$\nu^* = \min_{\Omega \subseteq V} \frac{\text{Cut}(\Omega)}{D(\Omega)} = \min_{F \subseteq E} \frac{\text{Cut}(F)}{D(F)}. \tag{37}$$

B. Dual of the Steiner flow problem

We would like to show provable bounds on the ratio between γ^* (Steiner packing rate) and ν^* (the cut). This can then be translated into bounds on the ratio between α (computation rate) and ν^* (the cut). In order to do this, we first write the dual of the linear program for γ^*. By strong duality, the optimal value of the linear program is equal to the optimal value of the dual program. In the dual program, there is a dual variable y_k for each k, corresponding to the constraints (31) in the primal and a dual variable ℓ_e for each $e \in E$ corresponding to the constraints (32) in the primal. For each primal variable f_τ, we write a constraint (39) and there is a single constraint (40). We note that we can treat the graph as fully connected without loss of generality (since the capacity of non-existent edges can be set to zero). The dual can be written as follows:

$$\begin{aligned} \gamma^* &= \min \sum_{e \in E} c_e \ell_e \quad \text{s.t.} && (38) \\ \sum_{e: e \in \tau} \ell_e &\geq y_k \quad \forall \tau \in \mathcal{T}_k \ \forall k \in [K] && (39) \\ \sum_{k \in [K]} D_k y_k &\geq 1. && (40) \end{aligned}$$

Let $w_\ell(\tau)$ be the weight of the tree τ with weights ℓ, i.e., edge e has weight ℓ_e,

$$w_\ell(\tau) = \sum_{e: e \in \tau} \ell_e. \tag{41}$$

Furthermore we define $w_\ell(G_k)$ as the minimum weight of the Steiner tree with nodes G_k:

$$w_\ell(G_k) = \min_{\tau \in \mathcal{T}_k} w_\ell(\tau). \tag{42}$$

With this notation the y_k in the optimization problem is set equal to $w_\ell(G_k)$ and therefore can be rewritten as follows:

$$\begin{aligned} \gamma^* &= \min \sum_{e \in E} c_e \ell_e \quad \text{s.t.} \\ \sum_{k \in [K]} D_k w_\ell(G_k) &\geq 1. && (43) \end{aligned}$$

C. Tree networks

We will first consider the case when the network graph is an undirected tree. We would like to show that $g(S, K) = 1$. Since the graph is a tree, there is only one Steiner tree τ_k for each set G_k, i.e.,

$$\mathcal{T}_k = \{\tau_k\} \quad \forall k \in [K]. \tag{44}$$

Therefore,

$$w_\ell(G_k) = w_\ell(\tau_k) \quad \forall k \in [K]. \tag{45}$$

Thus the dual program for Steiner tree packing can be rewritten as

$$\begin{aligned} \gamma^* &= \min \sum_{e \in E} c_e \ell_e \quad \text{s.t.} \\ \sum_{k \in [K]} D_k w_\ell(\tau_k) &\geq 1. && (46) \end{aligned}$$

We would like to compare the optimal value of this program to the sparsest cut. We start with the optimal solution $\ell(e)\ \forall e \in E$ for this dual program and then try to obtain a cut Ω whose sparsity $\nu(\Omega)$ is close to the value of this dual program.

When the graph is a tree, we show that there is a sparsest cut among the cuts that remove a single edge. For an edge e, we define $D(e)$ as the total sum of demands separated by removing the edge e (similar to the definition of $D(\Omega)$). We write $e|G_k$ to denote that edge e disconnects at least one node of G_k from another node of G_k. Thus

$$D(e) = \sum_{k \in [K]} \mathbb{1}_{[e|G_k]} D_k. \tag{47}$$

In this notation we can also write the Steiner tree τ_k for G_k as

$$\tau_k = \{e : e|G_k\} \quad \forall k \in [K], \tag{48}$$

since τ_k includes every edge that separates G_k.

We start with the optimal dual variables $\ell(e), e \in E$, which is feasible for the dual program.

$$\begin{aligned} \nu^* &= \min_{F \subseteq E} \frac{\text{Cut}(F)}{D(F)} \leq \min_{e \in E} \frac{c(e)}{D(e)} = \min_{e \in E} \frac{\ell_e c(e)}{\ell_e D(e)} \\ &\overset{(a)}{\leq} \frac{\sum_{e \in E} \ell_e c(e)}{\sum_{e \in E} \ell_e D(e)} \\ &= \frac{\sum_{e \in E} \ell_e c(e)}{\sum_{e \in E} \ell_e \sum_{k \in [K]} \mathbb{1}_{[e|G_k]} D_k} \\ &\overset{(b)}{=} \frac{\sum_{e \in E} \ell_e c(e)}{\sum_{k \in [K]} D_k \{\sum_{e \in \tau_k} \ell_e\}} \\ &= \frac{\sum_{e \in E} \ell_e c(e)}{\sum_{k \in [K]} D_k w_\ell(\tau_k)} \\ &\overset{(c)}{\leq} \sum_{e \in E} \ell_e c(e) \\ &\overset{(d)}{=} \gamma^*, \end{aligned}$$

where (a) follows due to the standard inequality

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i}, \tag{49}$$

(b) follows from (48), the inequality (c) follows because ℓ is feasible for the dual program and hence satisfies the constraints in (46),

$$\sum_{k \in [K]} D_k w_\ell(\tau_k) \geq 1, \tag{50}$$

and (d) is immediate from (46) as well.

Since, $\gamma^* \leq \nu^*$ always, we have that $\gamma^* = \nu^*$ and the value of the sparsest cut is equal to the maximum concurrent Steiner flow, if the graph is a tree.

D. General Network

Now, we move on to considering a general network. For a general network, the following result is known [2]:

$$\frac{1}{O(\log^2 SK)}\nu^* \leq \gamma^* \leq \nu^*, \tag{51}$$

where the function $O(\log^2 SK)$ does not depend on the number of nodes, capacity of edges in the network or the demands $D_1, ..., D_K$. The approximation factor was refined to $O(\log n)$ in [3] (see Section 3.1 there) and then to $O(\log |\cup_k G_k|)$ in [4]. Note that $|\cup_k G_k| \leq SK$. Therefore, we can write,

$$\frac{1}{\kappa(\log SK)}\nu^* \leq \gamma^* \leq \nu^*, \tag{52}$$

for some universal constant κ.

This result is implicit in [3] and [4], and we refer the reader there for the proof of this result. The basic idea of the proof is to connect Steiner flows and cuts in general networks to Steiner flows and cuts in tree networks. This is done using the notion of embeddings of general metric spaces into "tree metrics".

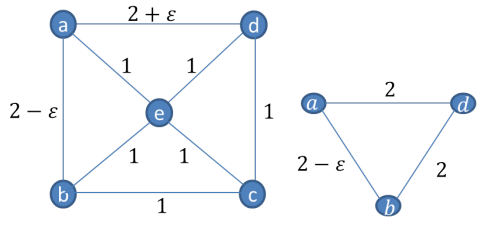
Now, we can use the relationship between Steiner packing and cuts in order to derive approximations for function computation capacity. Using (19), we get $\alpha = \frac{\gamma^*}{\lambda_f}$. This implies that, for computation of λ_f-divisible functions,

$$\frac{1}{\lambda_f \kappa(\log SK)}\nu^* \leq \alpha. \tag{53}$$

Since this approximation factor $O(\log SK)$ between the achievable rate and the cut does not depend on the demand vector $D_1, ..., D_K$, we can show that this approximation factor holds for the entire rate region,

$$\frac{1}{\lambda_f \kappa(\log SK)}\bar{\mathcal{C}} \subseteq \mathcal{R}_{\text{comp}}, \tag{54}$$

which proves the achievable portion of the result.



(a) A Weighted Network (b) Reduced Graph

Fig. 4: Restricted Steiner Trees

1) Algorithm for finding Steiner packings: First, we observe that the Steiner tree packing problem in general graphs (even for the case of a single session) is NP-hard [7]. We will describe a polynomial time algorithm, proposed in [2], for computing Steiner tree packing whose packing number is within a factor of two of the optimal Steiner tree packing. The *ellipsoid method* [7] gives a way of converting optimization problems into feasibility checking. In particular, given an assignment of the variables, if there is a polynomial time algorithm (called the *separation oracle*) that will either say it is feasible or produce a separating hyperplane that separates the feasible set and the assigned variables, then we can use this separation oracle to do optimization in polynomial time as well.

We will apply the ellipsoid method to the dual program of Steiner tree packing as formulated in (43). Given an assignment of $\ell(e)$, checking the feasibility is very easy once we compute $w_\ell(G_k)$, i.e., the minimum weight of the Steiner tree with nodes G_k. This minimum weight Steiner tree problem is NP-hard , however there is a factor 2 approximation algorithm that runs in polynomial time [7]. We briefly describe this algorithm. Given a set of distances $\ell(e)$, create a graph H_k, which has vertices G_k and is fully connected. The distance between two nodes u and v in H_k is the distance of the shortest path P_{uv} between u and v in the original graph G with distance $\ell(e)$ on edge $e = uv$. Now, we find a minimum weight spanning tree with edges E_k in H_k using Prim's or Kruskal's algorithm [8]. This yields a Steiner tree τ_k with edge set $\cup_{uv:(uv)\in E_k} P_{uv}$ on the original graph G (we may have to delete certain edges to get a tree). Steiner trees obtained in this manner are called restricted Steiner trees. It can be shown that the weight of the restricted Steiner tree, denoted by $\tilde{w}_\ell(G_k)$, is within a factor 2 of the minimum weight Steiner tree, i.e.,

$$w_\ell(G_k) \leq \tilde{w}_\ell(G_k) \leq 2w_\ell(G_k). \tag{55}$$

For an example of the minimum weight Steiner tree problem, see Fig. 4. A weighted graph is shown in Fig. 4a, where we would like to construct a minimum weight Steiner tree with nodes a,b, and d. To do so, we construct a graph H on nodes a,b and d with weights equal to the minimum distance between these nodes in the original graph. This graph is shown in Fig. 4b. The minimum weight spanning tree in H is given by the edges ba and ad of weight $4 - \epsilon$. This corresponds to a tree in the original graph with the edges ba, ae and ed. The minimum weight Steiner tree in the original graph is of weight 3, given by the edges be, ae and ed.

Since the objective function is linear, using this approximation algorithm for minimum weight Steiner tree as the separation oracle in ellipsoid method yields a packing which is within a factor 2 of the best packing.

VII. Conclusions

In this paper, we studied computation of multiple functions of independent data in undirected graphs. We proposed a simple strategy for computation, based on packing Steiner trees and performing in-network computation along the Steiner tree. We showed that for a wide class of functions, the achievable strategy and the proposed outer bound are close by showing an approximation factor which is the product of λ_f and a logarithmic in the number of nodes involved in the computation.

For functions which have a large λ_f, this strategy is clearly not optimal. A large λ_f implies that different links in the computation tree use widely varying amounts of capacity. However, in our analysis and algorithms, we have only dealt with unit capacity computation trees (Steiner trees) and therefore, we take a performance hit. One possible research direction

is to model the varying amounts of information conveyed by the different links in a computation tree. We can then design new algorithms for packing Steiner trees, which have differing capacity constraints on distinct links. In particular, we can use certain sub-modularity properties and the structure of the function in order to both design these algorithms and to obtain tighter gaps between these polymatroidal Steiner packings and generalized cut bounds.

ACKNOWLEDGMENTS

The authors would like to thank Changho Suh for suggesting the problem of function computation in undirected graphs. The authors would like to thank Chandra Chekuri for several discussions and pointers to the algorithmic literature.

REFERENCES

[1] S. Kannan, C. Suh, and P. Viswanath, "Multi-terminal function computation in undirected graphs," *submitted to ISIT 2012*.

[2] P. N. Klein, S. A. Plotkin, S. Rao, and E. Tardos, "Approximation Algorithms for Steiner and Directed Multicuts," *J. Algorithms* vol. 22, no.2, pp. 241-269, 1997.

[3] V. Nagarajan and R. Ravi, "Approximation Algorithms for requirement cut on graphs," *Algorithmica* vol. 56, no.2, pp.198-213, 2010.

[4] A. Gupta, V. Nagarajan and R. Ravi, "An improved approximation algorithms for requirement cut," *Operations Research Letters* vol. 38, pp.322-325, 2010.

[5] J. Fackaroenphol, S. Rao and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," *Journal of Computer and System Sciences* vol. 69, no. 3, pp.485-497, 2004.

[6] C. Chekuri "Approximation Algorithms" *Course lecture notes* Available online: http://www.cs.illinois.edu/class/sp11/cs598csc/

[7] D. P. Williamson and D. B. Shmoys, "The design of approximation algorithms" *Cambridge University Press* 2011.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to algorithms" *MIT Press* 2001.

[9] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing: Cut-set bounds", *IEEE Trans. Information Theory*, vol. 57, no.2, pp. 1015 - 1030, Feb. 2011.

[10] H. Kowshik and P. R. Kumar, "Optimal Function Computation in Directed and Undirected Graphs," Available online: http://arxiv.org/abs/1105.0240

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

[12] F. T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM* vol. 46, No.6, pp. 787-832, 1999.

[13] Z. Li and B. Li. "Network Coding: The Case of Multiple Unicast Sessions," *Proc. 42nd Allerton Conference,* Monticello, IL, 2004.

[14] W. T. Tutte. "On the Problem of Decomposing a Graph Into n Connected Factors," *J. London Math. Soc.*, vol. 36, pp. 221230, 1961.

[15] C. St. J. A. Nash-Williams, "Edge-disjoint Spanning Trees of Finite Graphs," *J. London Math. Soc.,* vol. 36, pp. 445450, 1961.

[16] A. Frank, "Edge-connection of Graphs, Digraphs, and Hypergraphs," *Tech.Rep., Enervary Research Group on Combinatorial Optimization*, Budapest, Hungary, http://www.cs.elte.hu/egres, September 2000.

[17] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Linear Codes, Target Function Classes, and Network Computing Capacity", *Available online:* http://arxiv.org/abs/1102.4825

[18] C. Suh, N. Goela, and M. Gastpar, "Computation in Multicast Networks: Function Alignment and Converse Theorems," *Available online:* http://arxiv.org/abs/1209.3358

[19] A. C. Yao, "Some complexity questions related to distributive computing," in *Proceedings of the eleventh annual ACM Symposium on Theory of Computing*, 1979, pp. 209–213.

[20] E. Kushilevitz and N. Nisan, *Communication Complexity*. Cambridge University Press, 1997.

[21] A. Orlitsky and J. R. Roche. Coding for computing. *IEEE Transactions on Information Theory*, 47:903–917, 2001.

[22] N. Ma, P. Ishwar, and P. Gupta, "Information-theoretic bounds for multiround function computation in collocated networks," in *Proceedings of the IEEE International Symposium on Information Theory*, 2009, pp. 2306–2310.

[23] D. Mosk-Aoyama and D. Shah, "Fast distributed algorithms for computing separable functions," *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 2997–3007, Jul. 2008.

[24] J. Chuzhoy and S. Khanna. "Polynomial Flow-Cut Gaps and Hardness of Directed Cut Problems," *JACM*, vol. 56, no. 2, 2009.

[25] V. Doshi, D. Shah, and M. Medard, "Source coding with distortion through graph coloring," *Proceedings of the IEEE International Symposium on Information Theory*, 2007, pp. 1501–1505.

[26] O. Ayaso, D. Shah, and M. Dahleh, "Lower bounds on information rates for distributed computation via noisy channels," in *Proceedings of the forty-fifth Allerton Conference on Computation, Communication and Control*, 2007.

[27] B. Nazer and M. Gastpar, "Computing over multiple-access channels," *IEEE Transactions on Information Theory*, vol. 53, pp. 3498–3516, Oct. 2007.

[28] U. Niesen, B. Nazer and P. Whitting. Computation Alignment: Capacity Approximation without Noise Accumulation. *Available Online,* http://arxiv.org/abs/1108.6312

[29] L. Ying, R. Srikant, and G. E. Dullerud. Distributed symmetric function computation in noisy wireless sensor networks. *IEEE Transactions on Information Theory*, 53(12):4826–4833, December 2007.

[30] C. Dutta, Y. Kanoria, D. Manjunath, and J. Radhakrishnan. A tight lower bound for parity in noisy communication networks. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1056–1065, January 2008.

[31] M. Charikar, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, "Approximation Algorithms for Directed Steiner Problems" *Journal of Algorithms*, vol. 33, pp. 73-91, 1999.

[32] A. Agarwal and M. Charikar, "On the advantage of Network Coding for Improving Network Throughput," *Proc. Information Theory Workshop*, 2004.

[33] B. K. Rai, B. K. Dey, and S. Shenvi, "Some bounds on the capacity of communicating the sum of sources," in *ITW 2010, Cairo*, 2010.

[34] A. Ramamoorthy, "Communicating the sum of sources over a network," in *Proceedings of the IEEE International Symposium on Information Theory*, 2008, pp. 1646–1650.

[35] M. Langberg and A. Ramamoorthy, "Communicating the sum of sources in a 3-sources/3-terminals network," in *Proceedings of the IEEE International Symposium on Information Theory*, 2009, pp. 2121–2125.

[36] B. K. Rai and B. K. Dey, "Feasible alphabets for communicating the sum of sources over a network," in *Proceedings of the IEEE International Symposium on Information Theory*, 2009, pp. 1353–1357.

[37] B. K. Rai, and B. K. Dey, "On Network Coding for Sum-Networks," *IEEE Transactions on Information Theory*, vol. 58, no. 1, pp. 50–63, 2012.

[38] S. Subramanian, P. Gupta, and S. Shakkottai, "Scaling bounds for function computation over large networks," in *Proceedings of the IEEE International Symposium on Information Theory*, 2007, pp. 136–140.

[39] V. Shah, B. Dey, and D. Manjunath, "Network Flows for Functions, " *Proc. ISIT 2011*, St. Petersburg, Russia, Aug. 2011.

[40] R. Ahlswede and Ning Cai. On communication complexity of vector-valued functions. *IEEE Transactions on Information Theory*, 40:2062–2067, 1994.

[41] A. Anandkumar, M. Wang, L. Tong, and A. Swami. Prize-Collecting Data Fusion for Cost-Performance Tradeoff in Distributed Inference. *Proc. of IEEE INFOCOM,* Rio De Janeiro, Brazil, Apr. 2009.