Upper Bound on Function Computation in Directed Acyclic Networks

Cupjin Huang, Zihan Tan and Shenghao Yang
Institute for Interdisciplinary Information Sciences
Tsinghua University, Beijing, China

Abstract—Function computation in directed acyclic networks is considered, where a sink node wants to compute a target function with the inputs generated at multiple source nodes. The network links are error-free but capacity-limited, and the intermediate network nodes perform network coding. The target function is required to be computed with zero error. The computing rate of a network code is measured by the average number of times that the target function can be computed for one use of the network. We propose a cut-set bound on the computing rate using an equivalence relation associated with the inputs of the target function. Our bound holds for general target functions and network topologies. We also show that our bound is tight for some special cases where the computing capacity can be characterized.

I. INTRODUCTION

We consider function computation in a directed acyclic network, where a target function \( f \) is intended to be calculated at a sink node, and the input symbols of the target function are generated at multiple source nodes. As a special case, network communication is just the computation of the identity function.\(^1\) Network function computation naturally arises in sensor networks [1] and Internet of Things, and may find applications in big data processing.

Various models and special cases of this problem have been studied in literature (see the summarizations in [2]–[4]). We are interested in the following network coding model for function computation. Specifically, we assume that the network links have limited (unit) capacity and are error-free. Each source node generates multiple input symbols, and the network codes perform vector network coding by using the network multiple times.\(^2\) An intermediate network node can transmit the output of a certain fixed function of the symbols it receives. Here all the intermediated nodes are considered with unbounded computing ability. The target function is required to be computed correctly for all possible inputs. We are interested in the computing rate of a network code that computes the target function, i.e., the average number of times that the target function can be computed for one use of the network. The maximum achievable computing rate is called the computing capacity.

When computing the identity function, the problem becomes the extensively studied network coding [5], [6], and it is known that in general linear network codes are sufficient to achieve the multicast capacity [6], [7]. For linear target functions over a finite field, a complete characterization of the computing capacity is not available for networks with one sink node. Certain necessary and sufficient conditions have been obtained such that linear network codes are sufficient to calculate a linear target function [4], [8]. But in general, linear network codes are not sufficient to achieve the computing capacity of linear target functions [9].

Networks with a single sink node are discussed in this paper, while both the target function and the network code can be non-linear. In this scenario, the computing capacity is known when the network is a multi-edge tree [2] or when the target function is the identity function. For the general case, various bounds on the computing capacity based on cut sets have been studied [2], [3].

We find, however, that the upper bounds claimed in the previous works do not generally hold. For an example that we will evaluate, the computing capacity is strictly larger than the two upper bounds claimed in [2], [3] respectively, where the issue is related to the classification of the inputs of the target function that are in a certain sense equivalent for each cut. Towards a general upper bound, we define an equivalence relation associated with the inputs of the target function (but does not depend on the network topology) and propose a cut-set bound on the computing capacity using this equivalence relation. Our bound holds for general target functions and general network topologies in the network coding model. We also show that our bound is tight when the network is a multi-edge tree or when the target function is the identity function.

In the remainder of this paper, Section II formally introduces the network computing model. The upper bound of the computing rate is given in Theorem 3, and is proved in Section IV. Section III compares with the previous results and discusses the tightness of our upper bound. Omitted proofs in this paper can be found in [10].

II. MAIN RESULTS

In this section, we will first introduce the network computing model. Then we will define cut sets and discuss some special cases of the function computation problem. Last we head to the main theorem about the cut-set bound for function computation.

A. Function-Computing Network Codes

Let \( G = (\mathcal{V}, \mathcal{E}) \) be a directed acyclic graph (DAG) with a finite vertex set \( \mathcal{V} \) and an edge set \( \mathcal{E} \), where multi-edges between a certain pair of nodes are allowed. A network over \( G \) is denoted as \( \mathcal{N} = (G, S, \rho) \), where \( S \subset \mathcal{V} \) is called the source nodes and \( \rho \in \mathcal{V} \setminus S \) is called the sink node \( \rho \). Let \( s = |S| \),
and without loss of generality (WLOG), let \( S = \{1, 2, \ldots, s\} \).

For an edge \( e = (u, v) \), we call \( u \) the tail of \( e \) (denoted by \( \text{tail}(e) \)) and \( v \) the head of \( e \) (denoted by \( \text{head}(e) \)). Moreover, for each node \( u \in V \), let \( \mathcal{E}(u) = \{ e \in \mathcal{E} : \text{head}(e) = u \} \) and \( \mathcal{E}_0(u) = \{ e \in \mathcal{E} : \text{tail}(e) = u \} \) be the set of incoming edges and the set of outgoing edges of \( u \), respectively. Fix a topological order of the vertex set \( V \). This order naturally induces an order of the edge set \( \mathcal{E} \), where edges \( e > e' \) if either \( \text{tail}(e) > \text{tail}(e') \) or \( \text{tail}(e) = \text{tail}(e') \) and \( \text{head}(e) > \text{head}(e') \). WLOG, we assume that \( \mathcal{E}(j) = \emptyset \) for all source nodes \( j \in S \), and \( \mathcal{E}_0(\rho) = \emptyset \). We will illustrate in Section III-C how to apply our results on a network with \( \mathcal{E}(j) \neq \emptyset \) for certain \( j \in S \).

The network defined above is used to compute a function, where multiple inputs are generated at the source nodes and the output of the function is demanded by the sink node. The computation units with unbounded computing ability are allocated at all the network nodes. However, the computing capability of the network will be bounded by the network transmission capability. Denote by \( B \) a finite alphabet. We assume that each edge can transmit a symbol in \( B \) reliably for each use.

Denote by \( A \) and \( O \) two finite alphabets. Let \( f : A^s \to O \) be the target function, which is the function to be computed via the network and whose \( i \)-th input is generated at the \( i \)-th source node. We may use the network to compute the function multiple times. Suppose that the \( j \)-th source node consecutively generates \( k \) symbols in \( A \) denoted by \( x_{1j}, x_{2j}, \ldots, x_{kj} \), and the symbols generated by all the source nodes can be given as a matrix \( x = (x_{ij})_{k \times s} \). Denote by \( x_j \) the \( j \)-th column of \( x \), and denote by \( x^i \) the \( i \)-th row of \( x \). In other words, \( x_j \) is the vector of the symbols generated at the \( j \)-th source node, and \( x^i \) is the input vector of the \( i \)-th computation of the function \( f \). Define for \( x \in A^{k \times s} \)

\[
f^{(k)}(x) = (f(x^1), f(x^2), \ldots, f(x^k))^\top.
\]

For convenience, we denote by \( x_j \) the submatrix of \( x \) formed by the columns indexed by \( J \subseteq S \), and by \( x^i \) the submatrix of \( x \) formed by the rows indexed by \( I \subseteq \{1, 2, \ldots, k\} \). We equate \( A^{I \times s} \) with \( A^s \) in this paper.

For two positive integers \( n \) and \( k \), a \((n, k)\) (function-computing) network code over network \( \mathcal{N} \) with target function \( f \) is defined as follows. Let \( x \in A^{k \times s} \) be the matrix formed by symbols generated at the source nodes. The purpose of the code is to compute \( f^{(k)}(x) \) by transmitting at most \( n \) symbols in \( A \) on each edge in \( E \). Denote the symbols transmitted on edge \( e \) by \( g^e(x) \in B^n \). For a set of edges \( E \subset \mathcal{E} \) we define

\[
g^E(x) = (g^e(x)|e \in E)
\]

where \( g^{e_1}(x) \) comes before \( g^{e_2}(x) \) whenever \( e_1 < e_2 \). The \((n, k)\) network code contains the encoding function for each edge \( e \), define:

\[
h^e : \begin{cases} A^k \to B^n, & \text{if } u \in S; \\ \prod_{e' \in \mathcal{E}(\text{tail}(e))} B^n \to B^n, & \text{otherwise.} \end{cases}
\]

Functions \( h^e, e \in \mathcal{E} \) determine the symbols transmitted on the edges. Specifically, if \( e \) is an outgoing edge of the \( i \)-th source node, then

\[
g^e(x) = h^e(x_i);
\]

if \( e \) is an outgoing edge of \( u \in V \setminus (S \cup \{\rho\}) \), then

\[
g^e(x) = h^e \left( g^{\mathcal{E}(u)}(x) \right).
\]

The \((n, k)\) network code also contains a decoding function

\[
\varphi : \prod_{e \in \mathcal{E}(\rho)} B^n \to O^k.
\]

Define

\[
\psi(x) = \varphi \left( g^{\mathcal{E}(\rho)}(x) \right).
\]

If the network code computes \( f \), i.e., \( \psi(x) = f^{(k)}(x) \) for all \( x \in A^{k \times s} \), we then call \( \frac{k}{n} \log_{|A|} |A| \) an achievable computing rate, where we multiply \( \frac{k}{n} \) by \( \log_{|A|} |A| \) in order to normalize the computing rate for target functions with different input alphabets. The computing capacity of network \( \mathcal{N} \) with respect to a target function \( f \) is defined as

\[
C(\mathcal{N}, f) = \sup \left\{ \frac{k}{n} \log_{|A|} |A| \left| \frac{k}{n} \log_{|A|} |A| \text{ is achievable} \right. \right\}.
\]

B. Cut Sets and Special Cases

For two nodes \( u \) and \( v \) in \( V \), denote the relation \( u \to v \) if there exists a directed path from \( u \) to \( v \) in \( G \). If there is no directed path from \( u \) to \( v \), we say \( u \) is separated from \( v \). Given a set of edges \( C \subset E \), \( IC \) is defined to be the set of source nodes which are separated from the sink node \( \rho \) if \( C \) is deleted from \( \mathcal{E} \). Set \( C \) is called a cut set if \( IC \neq \emptyset \), and the family of all cut sets in network \( \mathcal{N} \) is denoted as \( \Lambda(\mathcal{N}) \).

Additionally, we define the set \( KC \) as

\[
KC = \{ i \in S \mid \exists v, t \in V, i \to v, (v, t) \in C \}.
\]

It is reasonable to assume that \( u \to \rho \) for all \( u \in V \). Then one can easily see that \( KC \) is the set of source nodes from which there exists a path to the sink node through \( C \). Define \( JC = KC \setminus IC \).

The problem also becomes simple when \( s = 1 \).

**Proposition 1.** For a network \( \mathcal{N} \) with a single source node and any target function \( f : A \to O \),

\[
C(\mathcal{N}, f) = \min_{C \in \Lambda(\mathcal{N})} \frac{|C|}{\log_{|A|} |f[A]|}.
\]

**C. Upper Bounds**

In this paper, we are interested in the general upper bound on \( C(\mathcal{N}, f) \). The first upper bound is induced by Proposition 1.

**Proposition 2.** For a network \( \mathcal{N} \) with target function \( f \),

\[
C(\mathcal{N}, f) \leq \min_{C \in \Lambda(\mathcal{N}) : |V_0| = s} \frac{|C|}{\log_{|A|} |f[A]|}.
\]

**Proof:** Build a network \( \mathcal{N}' \) by joining all the source nodes of \( \mathcal{N} \) into a single “super” source node. Since a code for network \( \mathcal{N} \) can be naturally converted to a code for network
By Theorem 3, we have
\[ C(N,f) \leq C(N',f). \]

The proof is completed by applying Proposition 1 on \( N' \) and \( \Lambda(N') = \{ C \in \Lambda(N) : I_C = S \} \).

The above upper bound only uses the image of function \( f \). We propose an enhanced upper bound by investigating an equivalence relation on the input vectors of \( f \). We will compare this equivalence relation with similar definitions proposed in [2], [3] in the next section.

**Definition 1** (Equivalence Class). For any function \( f : A^s \to \mathcal{O} \), any two disjoint index sets \( I, J \subseteq S \), and any \( a, b \in A^{|I|}, c \in A^{|J|} \), we say \( a \equiv b \mod I, J \) if for every \( x, y \in A^{|I|\times s} \), we have \( f(x) = f(y) \) whenever \( x_I = a, y_I = b, x_J = y_J = c \) and \( x_{S \setminus (I \cup J)} = y_{S \setminus (I \cup J)} \). Two vectors \( a \) and \( b \) satisfying \( a \equiv b \mod I, J \) are said to be \( (I, J, c) \)-equivalent. When \( J = \emptyset \) in the above definition, we use the convention that \( c \) is an empty matrix.

Note that the definition of equivalence does not require a previously given network. However, it will soon be clear that with a network, the division of equivalence classes naturally leads to an upper bound of the network function-computing capacity based on cut sets.

For every \( f, I, J \) and \( c \in A^{|J|} \), let \( W_{I \to J}^{(c)} \) denote the total number of equivalence class induced by \( \equiv \) \( I, J \). Given a network \( N \) and a cut set \( C \), let \( W_{C,f} = \max_{c \in A^{|J|}} W_{I \to J}^{(c)} \). Our main result is stated as following. The proof of the theorem is presented in Section IV).

**Theorem 3.** If \( N \) is a network and \( f \) is a target function, then
\[ C(N,f) \leq \min_{C \in \Lambda(N)} \frac{|C|}{\log|A| W_{C,f}} := \text{min-cut}(N,f). \]

**III. DISCUSSION OF UPPER BOUND**

In this section, we first give an example to illustrate the upper bound. We compare our result with the existing ones, and proceed by a discussion about the tightness of the bound.

**A. An Illustration of the Bound**

First we give an example to illustrate our result. Consider the network \( N_1 \) in Fig. 1 with the object function \( f(x_1, x_2, x_3) = x_1x_2 + x_3 \), where \( A = B = O = \{0, 1\} \).

Let us first compare the upper bounds in Theorem 3 and Proposition 2. Let \( C_0 = \{ e_6, e_7 \} \). Here we have

- \(|C_0| = 2, I_{C_0} = (3), J_{C_0} = (1, 2) \); and
- For any given inputs of nodes 1 and 2, different inputs from node 3 generate different outputs of \( f \). Therefore \( W_{I_{C_0} \to J_{C_0}, f}^{(c)} = 2 \) for any \( c \in A^2 \) and hence \( W_{C_0,f} = 2 \).

By Theorem 3, we have
\[ C(N_1,f) \leq \text{min-cut}(N_1,f) \leq \frac{|C_0|}{\log|A| W_{C_0,f}} = 2. \]

**B. Comparison with Previous Works**

Upper bounds on the computing capacity have been studied in [2], [3] based on a special case of the equivalence class defined in Definition 1. However, we will demonstrate that the bounds therein do not hold for the example we studied in the last subsection.

In Definition 1, when \( J = \emptyset \), we will say \( a \equiv b \mod I \), or \( a \) and \( b \) are \( I \)-equivalent. That is \( a \equiv b \mod I \) if for every \( x, y \in A^{|I|} \), we have \( f(x) = f(y) \).

**Fig. 1.** Network \( N_1 \) has three source nodes, 1, 2 and 3, and one sink node \( \rho \) that computes the nonlinear function \( f(x_1, x_2, x_3) = x_1x_2 + x_3 \), where \( A = B = O = \{0, 1\} \).

While Proposition 2 implies that
\[ C(N_1,f) \leq \min_{C \in \Lambda(N_1): I_C = S} \frac{|C|}{\log|A| |f(A^s)|} \]
\[ = \min_{C \in \Lambda(N_1): I_C = S} \frac{|C|}{|C|} = 4, \]
where the first equality follows from \( f(A^s) = \{0, 1\} \), and the second equality follows from
\[ \min_{C \in \Lambda(N_1): I_C = S} |C| = \{e_4, e_5, e_6, e_7\} = 4. \]

Therefore, Theorem 3 gives a strictly better upper bound than Proposition 2.

The upper bound in Theorem 3 is actually tight. We claim that there exists a \((1,2)\) network code that computes \( f \) in \( N_1 \). Consider an input matrix \( x = (x_i)_{i=1}^{2 \times 3} \). Node \( i \) sends \( x_{i1} \) to node \( \nu \) and sends \( x_{i3} \) to node \( \rho \) for \( i = 1, 2, 3 \). Node \( \nu \) computes \( f(x_1) = x_{11}x_{12} + x_{14} \) and sends it to node \( \rho \) via edge \( e_7 \). Node \( \rho \) receives its symbols from edges \( e_4, e_5 \) and \( e_6 \).

**Define**
\[ \text{min-cut}(N,f) = \min_{C \in \Lambda(N)} \frac{|C|}{\log|A| R_{C,f}}. \]
By Lemma 1, we have \( \text{min-cut}(\mathcal{N}, f) \geq \text{min-cut}_A(\mathcal{N}, f) \). It is claimed in [2, Theorem II.1] that \( \text{min-cut}_A(\mathcal{N}, f) \) is an upper bound on \( \mathcal{C}(\mathcal{N}, f) \). We find, however, \( \text{min-cut}_A(\mathcal{N}, f) \) is not universally an upper bound for the computing capacity. Consider the example in Fig. 1. For cut set \( C_1 = \{e_4, e_6, e_7\} \), we have \( I_{C_1} = \{1, 3\} \). On the other hand, it can be proved that \( R_{C_1,f} = 4 \) since \( f \) is an affine function of \( x_2 \) given that \( x_1 \) and \( x_3 \) are fixed, and it takes 2 bits to represent this affine function over the binary field. Hence

\[
\text{min-cut}_A(\mathcal{N}_1, f) \leq \frac{|C_1|}{\log_{|A|} R_{C_1,f}} = \frac{3}{2} < 2 = \mathcal{C}(\mathcal{N}_1, f).
\]

For a network \( \mathcal{N} \) as defined in Section II-A, we say a subset of nodes \( U \subset V \) is a cut if \( |U \cap S| > 0 \) and \( \rho \notin U \). For a cut \( U \), denote by \( \mathcal{E}(U) \) the cut set determined by \( U \), i.e.,

\[
\mathcal{E}(U) = \{e \in E : \text{tail}(e) \in V, \text{head}(e) \in V \setminus U\}.
\]

Let \( \Lambda^+(\mathcal{N}) = \{\mathcal{E}(U) : U \text{ is a cut in } \mathcal{N}\} \).

Define

\[
\text{min-cut}_K(\mathcal{N}, f) = \min_{C \in \Lambda^+(\mathcal{N})} \frac{|C|}{\log_{|A|} R_{C,f}}.
\]

Since \( \Lambda^+(\mathcal{N}) \subset \Lambda(\mathcal{N}) \), \( \text{min-cut}_A(\mathcal{N}, f) \geq \text{min-cut}_A(\mathcal{N}, f) \). It is implied by [3, Lemma 3] that \( \text{min-cut}_A(\mathcal{N}, f) \) is an upper bound on \( \mathcal{C}(\mathcal{N}, f) \). However, \( \text{min-cut}_A(\mathcal{N}, f) \) is also not universally an upper bound for the computing capacity. Consider the example in Fig. 1. For the cut \( U_1 = \{1, 3, v\} \), the corresponding cut set \( \mathcal{E}(U_1) = C_1 = \{e_4, e_6, e_7\} \). Hence,

\[
\text{min-cut}_K(\mathcal{N}_1, f) \leq \frac{|C_1|}{\log_{|A|} R_{C_1,f}} = \frac{3}{2} < 2 = \mathcal{C}(\mathcal{N}_1, f).
\]

C. Tightness

The upper bound in Theorem 3 is tight when the network is a multi-edge tree.

**Theorem 4.** If \( G \) is a multi-edge tree, for network \( \mathcal{N} = (G, S, \rho) \) and any target function \( f \),

\[
\mathcal{C}(\mathcal{N}, f) = \text{min-cut}(\mathcal{N}, f).
\]

The upper bound in Theorem 3 is not tight for certain cases. Consider the network \( \mathcal{N}_2 \) in Fig. 2(a) provided in [2]. Note that in \( \mathcal{N}_2 \), source nodes 1 and 2 have incoming edges. To match our model described in Section II-A, we can modify \( \mathcal{N}_2 \) to \( \mathcal{N}_2' \) shown in Fig. 2(b), where the number of edges from node \( i \) to node \( i' \) is infinity, \( i = 1, 2 \). Every network code in \( \mathcal{N}_2' \) naturally induces a network code in \( \mathcal{N}_2 \) and vice versa. Hence, we have

\[
\mathcal{C}(\mathcal{N}_2, f) = \mathcal{C}(\mathcal{N}_2', f).
\]

We then evaluate \( \text{min-cut}(\mathcal{N}_2', f) \). Note that

\[
\frac{|C|}{\log_{|A|} W_{C,f}} < \infty
\]

holds only if \( |C| < \infty \), and we can thus consider only the finite cut sets. For a finite cut set \( C \), we denote by \( C' = C \cap \{e_1, \ldots, e_4\} \). We have \( |C'| \leq |C| \) and \( J_{C'} \subseteq J_C \), and we claim \( I_{C'} = I_C \). Note that \( I_{C'} \subseteq I_C \). Suppose that there exists \( i \in I_C \setminus I_{C'} \), then there exists a path from \( i \) to \( \rho \) which is disjoint with \( C' \), but shares a subset \( D \) of edges with \( C \). Then \( D \subset \mathcal{E}_o(i) \) and hence \( |D| = 1 \). We simply replace the edge in \( D \) by an arbitrary edge in \( \mathcal{E}_o(i) \setminus C \) and form a new path from \( i \) to \( \rho \). This is always possible, since \( C \cap \mathcal{E}_o(i) \) is finite while \( \mathcal{E}_o(i) \) is not. The newly formed path is disjoint with \( C \), and then we have \( i \notin I_C \), a contradiction.

According to Lemma 1, we have \( W_{C,f} \geq W_{C,f} \) and hence

\[
\frac{|C|}{\log_{|A|} W_{C,f}} \leq \frac{|C|}{\log_{|A|} W_{C,f}}.
\]

Therefore we can consider only cut sets \( C' \subseteq \{e_1, e_2, e_3, e_4\} \). We then have \( \text{min-cut}(\mathcal{N}_2', f) = 1 \), where the minimum is obtained by the cut set \( \{e_2, e_4\} \). While for network \( \mathcal{N}_2 \), it has been proved in [2] that \( C(\mathcal{N}_2, f) = \log_6 4 < 1 \). Hence \( \text{min-cut}(\mathcal{N}_2', f) = 1 > C(\mathcal{N}_2', f) \).

IV. PROOF OF MAIN THEOREM

To prove Theorem (3), we first give the definition of F-extension and two lemmas.

**Definition 2.** [F-Extension] Given a network \( \mathcal{N} \) and a cut set \( C \in \Lambda(\mathcal{N}) \), define \( D(C) \subseteq E \) as

\[
D(C) = \bigcup_{i \notin I_C} \mathcal{E}_o(i).
\]

Then the F-extension of \( C \) is defined as

\[
F(C) = C \cup D(C).
\]

**Lemma 2.** For every cut set \( C \), \( F(C) \) is a global cut set, i.e.,

\[
\forall C \in \Lambda(\mathcal{N}), I_{F(C)} = S.
\]

**Proof:** Clearly, \( I_C \subseteq I_{F(C)} \), then it suffices to show that for all \( i \notin I_C \), we have \( i \notin I_{F(C)} \). This is true, since \( \mathcal{E}_o(i) \subseteq F(C) \) and \( i \in \mathcal{E}_o(i) \) imply \( i \notin F(C) \).

**Lemma 3.** Consider a \((n, k)\) network code in \( \mathcal{N} = (G, S, \rho) \). For any global cut set \( C \), \( \psi(x) \) is a function of \( g^C(x) \), i.e.,

\[
\psi(x) = \psi^{F(C)}(g^C(x))
\]

for certain function \( \psi^{F(C)} \).

**Proof:** For a global cut set \( C \) of \( \mathcal{N} \). Let \( G_C \) be the subgraph of \( G \) formed by the (largest) connected component.
of $G$ including $\rho$ after removing $C$ from $E$. Let $S_C$ be the set of nodes in $G_C$ that do not have incoming edges. Since $G_C$ is also a DAG, $S_C$ is not empty. For each node $u \in S_C$, we have i) $u$ is not a source node in $N$ since otherwise $C$ would not be a global cut set, and ii) all the incoming edges of $u$ in $G$ are in $C$ since otherwise $G_C$ can be larger. For each node $u$ in $G_C$ but not in $S_C$, the incoming edges of $u$ are either in $G_C$ or in $C$, since otherwise the cut set $C$ would not be global. If we can show that for any edge $e$ in $G_C$, $g^e$ is a function of $g^C(x)$, then $\psi(x) = \varphi(E(\rho))$ is a function of $g^C(x)$.

Suppose that $G_C$ has $K$ nodes. Consider the topological order on the set of nodes in $G_C$, and number these nodes as $u_1 < \ldots < u_K$, where $u_K = \rho$. Denote by $E_o(u|G_C)$ the set of outgoing edges of $u$ in $G_C$. We claim that $g^{E_o(u|G_C)}(x)$ is a function of $g^C(x)$ for $i = 1, \ldots, K$, which implies that for any edge $e$ in $G_C$, $g^e$ is a function of $g^C(x)$. We prove this inductively. First $g^{E_o(u|G_C)}(x)$ is a function of $g^C(x)$ since $u_1 \in S_C$ and hence all the incoming edges of $u_1$ in $G$ are in $C$. Assume that the claim holds for the first $k$ nodes in $G_C$, $k \geq 1$. For $u_k+1$, we have two cases: If $u_k+1 \in S_C$, the claim holds since all the incoming edges of $u_k+1$ in $G$ are in $C$. If $u_k+1 \notin S_C$, we know that $E_o(u_k+1) \subseteq E_o(u|G_C) \cup C$. By the induction hypothesis, we have that $g^{E_o(u|G_C)}(x)$ is a function of $g^C(x)$. The proof is completed.

In the following proof of Theorem 3, it will be handy to extend the equivalence relation for a block of function inputs. For disjoint sets $I, J \subseteq S$ and $c \in A^{1 \times |I|}$ we say $a, b \in A^{k \times |I|}$ are $(I, J, c)$-equivalent if for any $x, y \in A^{k \times |I|}$ with $x_I = a, y_I = b, x_J = c, y_J = \left(c^T, c^T, \ldots, c^T\right)^T$, and $x_S \in \cup_{i \in I} E_o(u_i|G_C)$, we have $f(x) = f(y)$. Then for the set $A^{k \times |I|}$, the number of equivalence classes induced by the equivalence relation is $\left(W^{(c)}_{I,J,f}\right)^k$.

**Proof of Theorem 3:** Suppose that we have a $(n, k)$ code with

$$\frac{k}{n} \log |B| |A| > \min \text{-cut}(N, f).$$

We show that this code cannot compute $f(x)$ correctly for all $x \in A^{k \times n}$. Denote

$$C^* = \arg \min_{C \subseteq A(N)} \frac{|C|}{\log |A| W_{C,f}}$$

and

$$c^* = \arg \max_{c \in A^{|I|}} W^{(c)}_{I,C^*,f}.$$

By (1)-(3), we have

$$\frac{k}{n} \log |B| |A| > \frac{|C^*|}{\log |A| W^{(c^*)}_{I,C^*,f}}.$$

which leads to

$$|B|^{C^*} n < \left(W^{(c^*)}_{I,C^*,f}\right)^k.$$  \hspace{1cm} (4)

Note that $g^{C^*}(x)$ only depends on $x_{K_C^*}$. By (4) and the pigeonhole principle, there exist $a, b \in A^{k \times |I|}$ such that $i$ and $b$ are not $(I_{C^*}, J_{C^*}, c^*)$-equivalent and ii) $g^{C^*}(x) = g^{C^*}(y)$ for any $x, y \in A^{k \times n}$ with

$$\begin{cases} x_{I_{C^*}} = a, & y_{I_{C^*}} = b, \\ x_{J_{C^*}} = y_{J_{C^*}} = \left(c^T, c^T, \ldots, c^T\right)^T, \\ x_{S \setminus K_{C^*}} = y_{S \setminus K_{C^*}}. \end{cases}$$

Fix $x, y \in A^{k \times n}$ satisfying (5) and $f^{(k)}(x) \neq f^{(k)}(y)$. The existence of such $x$ and $y$ is due to i). Since $C^*$ and $D(C^*)$ are disjoint (see Defintion 2) and for any $i \notin I_{C^*}$, $x_i = y_i$, together with ii), we have

$$g^{F^{(C^*)}}(x) = g^{F^{(C^*)}}(y).$$

Thus, applying Lemma 3 we have $\psi(x) = \psi(y)$. Therefore, the code cannot computes both $f^{(k)}(x)$ and $f^{(k)}(y)$ correctly. The proof is completed.

**V. CONCLUDING REMARKS**

We propose a new definition of equivalence classes associated with the inputs of a function, which enable us to obtain a general upper bound on the network function computing capacity. We show that the upper bound is tight when the network is a multi-edge tree.

**REFERENCES**


