

Javad Ebrahimi and Christina Fragouli

EPFL

Lausanne, Switzerland

Abstract

A long-standing open question in information theory is to characterize the unicast capacity of a wireless relay network. The difficulty arises due to the complex signal interactions induced in the network, since the wireless channel inherently broadcasts the signals and there is interference among transmissions. Recently, Avestimehr, Diggavi and Tse proposed a linear deterministic model that takes into account the shared nature of wireless channels, focusing on the signal interactions rather than the background noise. They generalized the min-cut max-flow theorem for graphs to networks of deterministic channels and proved that the capacity can be achieved using information theoretical tools. They showed that the value of the minimum cut is in this case the minimum rank of all the adjacency matrices describing source-destination cuts.

In this paper, we develop a polynomial time algorithm that discovers the relay encoding strategy to achieve the min-cut value in linear deterministic (wireless) networks, for the case of a unicast connection. Our algorithm crucially uses a notion of linear independence between channels to calculate the capacity in polynomial time. Moreover, we can achieve the capacity by using very simple one-symbol processing at the intermediate nodes, thereby constructively yielding finite length strategies that achieve the unicast capacity of the linear deterministic (wireless) relay network.

I. INTRODUCTION

Let $G = (V, E)$ denote a directed graph with unit capacity edges. We can think of each edge of this graph as a channel *orthogonal* to all other channels, where each channel (edge) has a single input and a single output, and can be used to send a single symbol from the input to the output (unit capacity). We can then depict a node with multiple incoming and outgoing edges as having multiple inputs and multiple outputs, as determined by its adjacent edges, where inputs and outputs can be arbitrarily connected to each other within the node. For example, Fig. 1(a) depicts a node in a directed graph, and Fig. 1(b) the equivalent representation of this node.

Wireless relay networks cannot be represented as graphs, due to the inherently shared nature of the wireless medium that causes complex signal interactions. In the wireless medium, transmissions are broadcasted, and may be received by multiple receivers at different signal strengths depending on path loss parameters. Moreover, there is interference between transmissions, and the signal from different nodes in the network can be received at very different power at a given receiver (high dynamic range of received signals). The characterization of the unicast capacity of a wireless relay network has been an open problem for decades, mainly due to these complex signal interactions.

Recently, Avestimehr, Diggavi and Tse [4], [5] proposed a linear deterministic network model (we will call this ADT model) that takes into account the interactions between the signals in a wireless network, i.e., broadcasting and interference, and represents the noise by a deterministic threshold rather than a random variable. The symbols received below the noise threshold are discarded. The argument is that for high Signal-to-Noise-Ratio (SNR), it is the signal interactions that will dominate the performance, and thus the capacity of the deterministic could be very close to that of the noisy network. Thus networks of deterministic channels could be used as approximate models for wireless networks.

The ADT model is based on the intuition of dividing the transmitted and received signals into symbols, where each symbol is transmitted at a different power level, and assuming that only symbols above a deterministic noise threshold will be successfully received. Deterministic networks can be over over an arbitrary field \mathbf{F}_q . In the following, when we do not explicitly specify the field, we will imply that the network operates over the binary field.

As an example, consider a point-to-point AWGN channel: $y = 2^{\alpha/2}x + z$, and assume that input bits x_1, x_2, \dots, x_n are transmitted from a node A , while a node B observes the signal y . The capacity is $\log(1 + 2^\alpha) \approx \alpha \log(2)$, assuming z is unit variance noise (α represents the channel gain in dB scale $\alpha \leftrightarrow \lceil \log(SNR) \rceil$). The ADT model over \mathbf{F}_2 in this case is obtained by truncating the received signal and assuming that the α most significant bits (MSB) of x are *always* above the deterministic noise threshold and received successfully at node B . The parameter α captures the path loss and determines how many of the MSB bits of x are received at y .

When broadcasting, each receiver node B_i will receive the m_i MSB from the transmitted bits x_1, x_2, \dots, x_n , with $0 \leq m_i \leq n$. For example, when in Fig. 2 node S transmits, node A_1 receives both the transmitted bits, while node A_2 receives only the MSB that was transmitted

with the higher power. The difference between the bit index at the transmitter and the bit index at the receiver represents path loss.

Interference in the ADT model is modeled through bit-wise binary addition, unlike Gaussian networks, where interfering signals are added through regular addition. In Fig. 2 the bit y_6 equals the binary addition (xor) of bits x_3 and x_4 . Again, the signal from different nodes in the network can be received at different power at a given receiver. For example, node D observes at y_9 the xor of x_5 and x_7 , i.e., the MSB from node B_1 and the 2^{nd} MSB from node B_2 . The generalization over an arbitrary field \mathbf{F}_q is straightforward, by substituting binary addition with addition over \mathbf{F}_q .

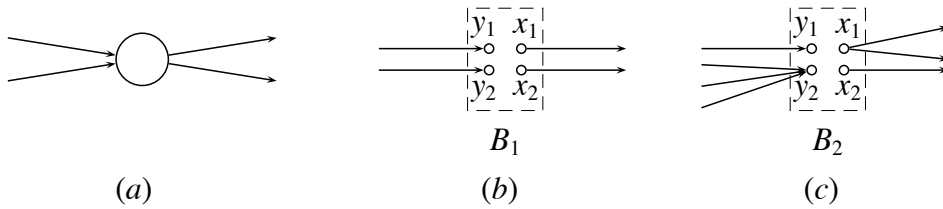


Fig. 1. (a) A node in a directed graph, (b) equivalent representation through orthogonal channels, and (c) a node in a network of deterministic channels.

In the ADT model, unlike graphs, channels are no longer orthogonal. Each input might be connected to multiple outputs belonging in different nodes, and the relationship between these inputs and outputs is determined by a set of linear equations. In Fig. 2, the channel between the nodes A_1, A_2 and B_1, B_2 can be described through the equations $y_6 = y_7 = x_3 + x_4$. A generic node of deterministic channel networks is depicted in Fig. 1(c). Loosely speaking, in deterministic networks, we can have Linear Dependence (LD) relationships between edges (we will make this precise in the following section), even though these edges might not be adjacent. For example, in Fig. 2, the edges (x_3, y_6) and (x_4, y_7) are linearly dependent. This makes challenging the task of calculating the min-cut value between a source-destination (S-D) pair and of identifying the node operations.

Avestimehr, Diggavi and Tse generalized the min-cut max-flow theorem for graphs to networks of deterministic channels and proved that the capacity can be achieved using information theoretical tools. They showed that the value of the minimum cut is in this case the minimum rank of all the adjacency matrices describing source-destination cuts. For example, in Fig. 2 the minimum cut value equals

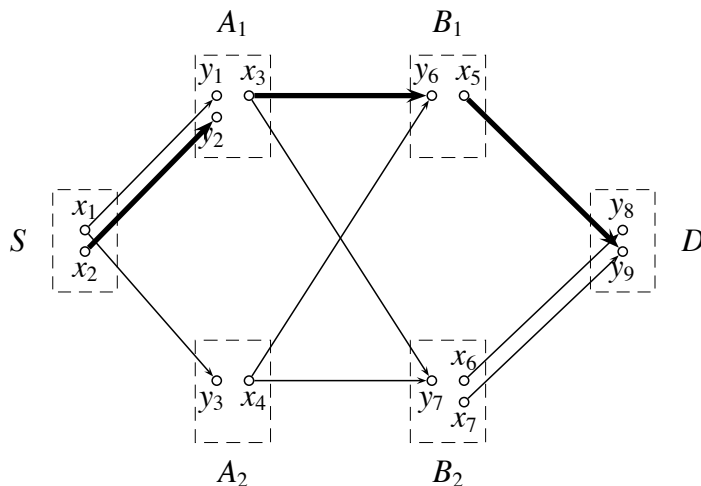


Fig. 2. An example of a linear binary deterministic network.

$$\text{rank} \begin{matrix} & y_6 & y_7 \\ x_3 & \begin{pmatrix} 1 & 1 \end{pmatrix} \\ x_4 & \begin{pmatrix} 1 & 1 \end{pmatrix} \end{matrix} = 1.$$

Note that there exists an exponential number of cuts, and thus identifying the capacity through exhaustive search becomes infeasible. In this paper, we develop a constructive polynomial-time algorithm which allows to efficiently calculate the min-cut value between a $S - D$ pair, and to achieve this value using simple operations at relay nodes.

To construct our algorithm, it is easy to see that, attempting to directly extend the Ford-Fulkerson (FF) algorithm [2], or other path-augmenting algorithms developed for graphs, is not straightforward. Indeed, assume that in Fig. 2 we have at a first iteration identified the path highlighted in bold. The FF algorithm may attempt to employ the path consisting of the edges (x_1, y_3) , (x_4, y_7) , (x_7, y_9) , which in fact is vertex disjoint (excluding the S, D nodes) from the already identified path. However, because edges (x_3, y_6) and (x_4, y_7) are LD, this path cannot bring innovative information to the destination; in fact, the min-cut value in this network equals one. Given that channels can interact in multiple ways, it is not clear that a polynomial algorithm does exist.

Even in regular graphs, the number of cuts between an $S-D$ pair is exponentially large. However, polynomial time algorithms do exist in that case. One way to understand this is

by observing that, in the FF algorithm for example, we are allowed to make “mistakes” when selecting a path, where a mistake in this case is when a path crosses a minimum cut more than once. The strength of the algorithm comes from the fact that such mistakes can be “corrected”, by allowing to use employed edges in the opposite direction. What these corrections do is effectively “rewiring” already identified partial-paths. For example in Fig. 3, a first iteration identifies the path that uses the edges AB , BC and DE . This path crosses a min-cut twice. A subsequent iteration can use edge CA in the opposite direction to find a new S-D path. This amounts to, no longer using edge BC and having two rewired paths: The first part of the first path arrives at node B, and is then complemented by the second path

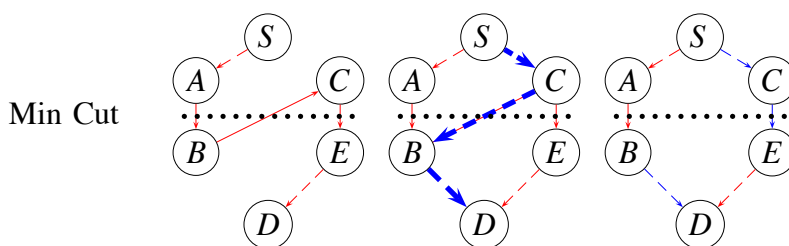


Fig. 3. Correcting a “mistake” in the Ford-Fulkerson algorithm can be thought of as “rewiring paths”.

from B to D. The second path arrives from S to E, and from E to D is complemented by the first path.

In deterministic networks, we cannot avoid making “mistakes” when selecting which paths to use, where now a mistake amounts to using the wrong edges between a set of linearly dependent edges; thus, to find a polynomial time algorithm, we need to put in place some simple mechanisms to “correct” such mistakes. As we will see in following sections, now using edges in opposite directions is no longer sufficient or helpful; we may in fact have to “jump” across nodes, and change the inputs or outputs employed by already identified paths. The interesting and surprising point is that, there exists a method to perform such corrections in polynomial time, and thus, no “mistake” is catastrophic.

We close this section by noting that in [4], it was observed that to study coding strategies and achievable rates, we can reduce an arbitrary network into a layered network, through a time-expansion technique, with asymptotically no rate-loss. Thus in this paper we will also focus our attention in layered networks, which will be defined formally in the next section.

This paper is based on the work in [11]. The algorithm in [11] was presented over binary fields. Moreover, the proof of the algorithm presented in [11] applies under some assumptions on the structure of the linear dependency between inputs and outputs. In this paper, we provide a simple modification of [11] that holds with no assumptions on the linear dependency of the channels. Moreover, we present the algorithm over an arbitrary finite field \mathbf{F}_q . The paper in [11] was followed up by a very nice connection with matroids and the development of alternative algorithms for this problem [12].

This paper is organized as follows. Section II introduces our notation and basic definitions. Section III describes our algorithm, provides a number of examples, and proves that it identifies a minimal value cut. Section IV concludes the paper.

II. MODEL AND DEFINITIONS

In this section, we start by defining the layered deterministic network model for a unicast connection over a network.

Definition 1: (Layered Deterministic Network). A layered deterministic network model over a finite field \mathbf{F}_q , consists of a set of nodes and a set of channels (or edges) with the following properties:

- 1) Each node consist of two sets, the set of inputs and the set of outputs of the node. We will generally denote inputs using the variable x and outputs using the variable y . We will denote by $A(x)$ and $A(y)$ respectively, the node where input x (output y) belongs to. Let I_{total} be the total number of inputs in the network and O_{total} the total number of outputs in the network.
- 2) The nodes of the deterministic model are partitioned in parts. Each part is called a layer of the network. We assume that each layer has at most M nodes, and denote by V_i the set of nodes in layer i . The layers are labeled by $i = 1, 2, \dots, \Lambda$, where Λ is the number of layers.
- 3) Layer 1 and layer Λ each has only one node in it. The node of the first layer is called “source node” and is denoted by S and the node of the last layer is called “receiver node” and is denoted by D . The source node has only outputs and the receiver node has only inputs.
- 4) Each channel is a link between an input of a node in layer i to an output of a node in layer $i + 1$ where $1 \leq i \leq \Lambda - 1$. A fixed nonzero value over a finite field \mathbf{F}_q is associated with each link.

- 5) Let \mathbf{x} denote a vector that collects all inputs in layer i , and \mathbf{y} a vector that collects all outputs in the next layer $i + 1$. Then these vectors are connected through a given linear transformation over \mathbf{F}_q , i.e., $\mathbf{y} = \mathbf{T}\mathbf{x}$, where each nonzero value in the transformation matrix \mathbf{T} corresponds to a channel and its associated value. \square

We can define a transformation matrix between an arbitrary subset of inputs and outputs in adjacent layers. Let V be a subset of all inputs in layer i and W be a subset of all outputs in layer $i + 1$ (for simplicity we do not include the i indices).

Definition 2: (Transfer Matrix). We define $\mathbf{T}(V, W)$ to be the matrix whose rows are labeled with the elements of V , the columns with the elements of W and the entry (v, w) is nonzero if and only if there is a channel between input v and output w . $\mathbf{T}(V, W)$ is called the transfer matrix between V and W . \square

We will describe the extension of a given transformation matrix $\mathbf{T}(V, W)$ by adding a row corresponding to an input $x \notin V$ as $\mathbf{T}(\{V, x\}, W)$ and the extension by adding both a row corresponding to an input x and a column corresponding to an output y (not already contained in V and W) as $\mathbf{T}(\{V, x\}, \{W, y\})$.

The maximum information S can send to D depends on the minimum cut value in the network, defined as follows.

Definition 3: (Cut and Cut-Value) By an $S - D$ cut \mathcal{V}_C we mean a partition of the nodes into two parts \mathcal{V}_1 and \mathcal{V}_2 in such a way that $S \in \mathcal{V}_1$ and $T \in \mathcal{V}_2$. We define the value of \mathcal{V}_C to be $\text{rank}\mathbf{T}(A_1, A_2) \log_2 q$, where rank refers to matrix rank, A_1 is the set of all inputs in the nodes in \mathcal{V}_1 , A_2 is the set of all outputs of the nodes in \mathcal{V}_2 , and q is the size of the employed finite field. The minimum cut value equals $\min_{\mathcal{V}_C} \text{rank}\mathbf{T}(A_1, A_2) \log_2 q$, where the minimization is over all $S - D$ cuts. \square

We will sometimes distinguish between a *layer-cut* and a *cross-cut*. There exist exactly $\Lambda - 1$ layer cuts, one between every two consecutive layers. For example, the j -layer cut is $\mathcal{V}_1 = V_1 \cup \dots \cup V_j$ and $\mathcal{V}_2 = V_{j+1} \cup \dots \cup V_\Lambda$ for $1 \leq j \leq \Lambda - 1$. A cross-cut involves several layers. The transfer matrix for a cross-cut is block diagonal, with the nodes in each layer belonging in a different block.

Next, we will define the notion of linear dependency between channels.

Definition 4: (LI and LD Channels). Suppose that H is a subset of channels between layers i and $i + 1$. Let V be the set of all inputs that are the head of a channel in H and W be the set of tails of these channels. We say H is a set of Linearly Independent (LI) channels if

$\text{rank}\mathbf{T}(V, W) = |H|$. Otherwise we say H is a set of Linearly Dependent (LD) channels. \square

Our algorithm will send information from S to D using $S - D$ paths, defined in the following. Through every path S sends one symbol over F_q to D .

Definition 5: ($S - D$ Path). An $S - D$ path is a disjoint set of edges $(e_1, e_2, \dots, e_{\Lambda-1})$ where e_1 starts from S , $e_{\Lambda-1}$ finishes at D , and e_i finishes at the same node where edge e_{i+1} starts. All $S - D$ paths have the same length $\Lambda - 1$, because of the structure of the layered network. \square

Essentially, selecting paths amounts to appropriately selecting sets of “used” inputs V and outputs W in each layer. To ensure that the information send through different paths can be decoded at the destination we need to use linearly independent (LI) paths, defined as follows.

Definition 6: (LI-Paths). Suppose that \mathcal{P} is a set of $S - D$ paths. We say these paths are linearly independent if and only if the set of edges of these paths in every layer form a set of linearly independent edges. \square

Note that each x and y can take part in at most one of the LI paths; in this case we will say that it is *used* by that path. That is, we will say that a channel input x is used, if there exists a path that uses a channel (x, y') for some y' . Similarly, we will say that a channel output y is used, if there exists a path that uses a channel (x', y) for some x' .

III. THE UNICAST ALGORITHM

A. Main idea

In our algorithm, we will find linearly independent paths one after another, in iterations. The first iteration identifies a path \mathcal{P}_1 . This is always possible if the source is connected to the destination, otherwise the capacity is zero. Each subsequent iteration identifies an additional path such that all selected paths are LI (as by definition 6). For example at iteration $K + 1$, the algorithm takes as input the LI paths $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_K\}$ and attempts to find path \mathcal{P}_{K+1} such that the paths $\{\mathcal{P}_1, \dots, \mathcal{P}_{K+1}\}$ are also LI (as by definition 6). Each iteration finishes once we reach the destination. The algorithm stops when an iteration cannot complete, at which point the algorithm outputs the set of identified LI paths \mathcal{P} .

To find a new path, we start from the source and select one channel at each layer until we reach the destination if possible. At each layer we need to select a valid channel, in the sense that it is linearly independent from the set of the channels of the identified paths in that layer in previous iterations. A main tool that we use to achieve this is that we allow the algorithm

to perform some type of “rewiring” inside one layer at a time. Assume for example that we have $K + 1$ “partial” paths from the source to nodes in layer i , and K “partial” paths from nodes from layer $i + 1$ to the destination. Rewiring refers to that we change the mapping between the starting and finishing paths by changing the channels we employ, while still preserving LI across the i -layer cut. These rewiring are achieved through two functions, the L -function and the ϕ -function, which we describe in detail later.

Note that, instead of selecting channels (or paths), we can equivalently think of our algorithm as appropriately selecting a subset of inputs and outputs to be used in each layer. Each node internally simply maps each of its used inputs to a used output (the specific mapping is not important). That is, selecting K paths amounts to selecting K inputs U_x at each layer i and K outputs U_y at the corresponding layer $i + 1$ such that the transfer matrix $\mathbf{T}_i = \mathbf{T}(U_x, U_y)$ is full rank for each i .

Each of the LI paths that the algorithm outputs will be used to convey an independent symbol over the field \mathbf{F}_q from the source to the destination. Let \mathbf{x} collect the K used outputs of the source and \mathbf{y} collect the K used inputs of the receiver. The overall transfer matrix $\mathbf{T} = \mathbf{T}_1 \cdot \mathbf{T}_2 \dots \mathbf{T}_{\Lambda-1}$ is full rank and therefore \mathbf{x} can be recovered at the receiver by solving the system of linear equations

$$\mathbf{y} = \mathbf{T}\mathbf{x} = \mathbf{T}_1 \cdot \mathbf{T}_2 \dots \mathbf{T}_{\Lambda-1}\mathbf{x}.$$

That is, although we send one symbol through each path, due to the linear combining the deterministic model imposes, the receiver will still need to solve equations to retrieve the data. By the choice of paths, that is, by selecting at each node the edges we use to collect and transmit information, we preserve the “degrees of freedom”, the number of independent linear equations the receiver decodes.

B. Algorithm Description

Assume we are at iteration $K + 1$, that takes as input the LI paths $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_K\}$ and attempts to find path \mathcal{P}_{K+1} such that the paths $\{\mathcal{P}_1, \dots, \mathcal{P}_{K+1}\}$ are also LI (as by definition 6).

During this iteration, we *explore* nodes, starting from the source node S . We will use the terminology of exploring a node A to indicate that we have found a path from S to A (LI from the paths in \mathcal{P}) and attempt to continue this path from node A to D in order to complete \mathcal{P}_{K+1} . Note that which input $y_i \in A$ we use to reach the node A does not play a

role; to explore a node it is sufficient that we arrive at it using any of its inputs. Once we reach a node, we *mark* the node as visited, and attempt to explore all edges emanating from it, as potential candidates for the path \mathcal{P}_{K+1} . We use an indicator variable \mathcal{M} with values $\{\text{T}, \text{F}\}$, to mark whether a node or edge has been explored (T) or not (F). We need explore (according to operations to be defined) each node during each iteration at most once, and we will do that calling a function E_A . Exploring a node reduces to exploring all unused inputs that it contains; exploring an input is achieved by calling a function E_x . Each edge may be explored during each iteration multiple times, for reasons we will explain in the following, but no more than a finite number of times. This ensures that each iteration terminates after a finite number of steps.

Assume that we have found a partial path \mathcal{P}_{K+1} from S to a node A in the i -layer and we explore input $x_i \in A$, with the goal of extending the path \mathcal{P}_{K+1} to the $i + 1$ -layer. Let $U = \{(x_i, y_j)\}$ denote the set of K used edges in the i -layer cut, U_x and U_y denote the set of used inputs and outputs respectively, and $\mathbf{T}(U_x, U_y)$ be the $K \times K$ full rank transformation matrix associated with U . We describe the steps we take to explore a specific input in the following. We illustrate these steps through a number of examples in Section III-C.

Steps in exploring input x_i at node A

- 1) If $x_i \in U_x$, i.e., x_i is already used by a path, do nothing. Note that although node A will be marked as explored ($\mathcal{M}(A) = \text{T}$), this particular $x_i \in A$ will not be marked ($\mathcal{M}(x_i) = \text{F}$ will remain).
- 2) If $x_i \notin U_x$, i.e., x_i is not used, then for each y_j , such that the channel (x_i, y_j) exists, we distinguish two cases.
 - a) $y_j \notin U_y$, i.e., y_j is not used. Consider the $(K+1) \times (K+1)$ matrix $\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})$ associated with the used edges and the new edge (x_i, y_j) . We again consider two cases.
 - i) If the matrix $\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})$ is not full rank, do nothing.
 - ii) If the matrix $\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})$ is full rank, use edge (x_i, y_j) to go to node $A(y_j)$. If this node has not been visited before, we attempt to continue from node $A(y_j)$ by calling the function $E_A(G, \mathcal{P}, \mathcal{M}, A(y_j))$.
Additionally, for each $y_k \in U_y$, with $A(y_k) = A(y_j)$, perform what we call the ϕ -function. The idea is that, in this case there exists a path from the source

to the destination identified during a previous iteration that goes through node $A(y_j)$. This path uses an edge $(x_k, y_k) \in U$ to reach node $A(y_j)$. We can then use our newly identified partial path that uses the edge (x_i, y_j) to reach from the source the node $A(y_j)$, and “connect” this new partial path with the existing partial path from $A(y_j)$ to destination. Thus, we have the opportunity to again perform rewirings and visit new nodes.

More precisely, the ϕ -function performs the following. Remove from the matrix $\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})$ the column corresponding to y_k with $A(y_k) = A(y_j)$. Let

$$\mathbf{C} \triangleq \mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\} - \{y_k\})$$

denote the resulting $(K + 1) \times K$ matrix. Consider each of the K square submatrices of \mathbf{C} resulting by deleting each of the first K rows. Let \mathbf{C}_m denote the submatrix resulting from deleting row x_m , i.e.,

$$\mathbf{C}_m \triangleq \mathbf{T}(\{U_x, x_i\} - \{x_m\}, \{U_y, y_j\} - \{y_k\}).$$

If \mathbf{C}_m is not full rank, do nothing. If it is full rank, perform a rewiring of the existing K paths using \mathbf{C}_m . If $A(x_m)$ is not marked as visited, explore $A(x_m)$. If $A(x_m)$ is marked as visited, then explore input x_m even if it is marked. Note that the ϕ -function may be executed at most as many times as the number of outputs in that layer, and thus when it is executed, at most K already visited inputs might be revisited. Examples 2-4 illustrate the use of the ϕ function.

- b) $y_j \in U_y$, i.e., y_j is used. We can then not immediately use the channel (x_i, y_j) , unless we perform some rewiring. This rewiring is captured by what we call the L_x -function. This function will be executed at most once for every input. To ensure that, we keep in the algorithm for each input an indicator variable \mathcal{ML} with values $\{T, F\}$.

The L_x -function operates as follows. Consider the extended transformation matrix $\mathbf{T}(\{U_x, x_i\}, U_y)$. Define $L_{x_i} \subseteq U_x$ to be the smallest subset of U_x , of size $|L_{x_i}| = s \leq K$, such that the matrix $\mathbf{T}(\{L_{x_i}, x_i\}, U_y)$ has rank s . Using proposition 2 this set can be identified in polynomial time. Proposition 3 proves that removing any one of the rows of $\mathbf{T}(\{L_{x_i}, x_i\}, U_y)$ still results in a full rank matrix. Equivalently,

removing any row of $\mathbf{T}(U_x, U_y)$ corresponding to a $x_k \in L_{x_i}$, and substituting it with the row corresponding to x_i , results in a full rank matrix, that can be used to rewire the paths identified in the previous iterations. That is, using Proposition 1, we can use the row $\mathbf{T}(x_i, U_y)$ to substitute any of the already employed $\mathbf{T}(x_k, U_y)$, $x_k \in L_{x_i}$ that are LD with x_i row, while still maintaining the same number of paths as identified from the previous iterations. We will then be left with a partial path arriving at the node $A(x_k)$, and we can attempt to use any of the available x 's in this node to proceed. We now distinguish to subcases:

- i) $A(x_k)$ is already marked as explored. In this case we will not visit this node again. However, we will explore input x_k , although this input might have been explored before. Note that, at each execution of the L_x function, at most K inputs will be re-examined.
- ii) $A(x_k)$ is not marked, i.e., during this iteration we visit this node for the first time. Then the algorithm explores this node. Additionally, if there exists a path identified during a previous iteration that utilizes (at the previous layer) an output y' at node $A(x_k)$ we will execute on this node the ϕ -function that we described previously.

Examples 1 and 4 illustrate the use of the L_x -function.

The previous steps are the main ingredients of two recursive functions E_A and E_X that implement our proposed algorithm and are summarized in Table I. The first function, E_x , checks if we can continue from a current input x to reach the destination by a sequence of channels which are linearly independent to the previous identified paths. The input of this function is the network, a family of identified paths and already visited nodes and current input. It returns true if there is a sequence of channels with the described properties and returns false, otherwise. The second function, E_A , does a similar job as the first function except that it works for the current node instead of the current channel. So, as one might guess, this function, essentially, calls the first function for all of its inputs and if none of them returns true, it also returns false. We illustrate the algorithm steps through a number of examples in Section III-C.

Three Propositions Used in the Algorithm

We here provide some useful propositions that were used in our algorithm. The first is a known property [9], that allows to “match” inputs and outputs through LI channels, and that that we repeat for completeness.

Proposition 1: If the $K \times K$ binary matrix $\mathbf{T}(U_x, U_y)$ is full rank, then there exist K LI edges with $x \in U_x$ and $y \in U_y$.

Proof: Since $\mathbf{T}(U_x, U_y)$ is full rank matrix, it has nonzero determinant. Now if we expand the determinant using the sum of product- expansion, we should have at least one non-zero product and this product corresponds to a perfect matching in the bipartite graph with adjacency matrix $\mathbf{T}(U_x, U_y)$. ■

Proposition 2: Let $\mathbf{T}(U_x, U_y)$ be a full rank $K \times K$ matrix and $\mathbf{x}_i \triangleq \mathbf{T}(x_i, U_y)$ a vector in its span. Then, we can find the smallest $L_{x_i} \subseteq U_x$ of size $s = |L_{x_i}| \leq K$ such that $\text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, U_y)) = \text{rank}(\mathbf{T}(L_{x_i}, U_y)) = s$ using $O(K^3)$ operations.

Proof: Since the matrix $\mathbf{A} \triangleq \mathbf{T}(U_x, U_y)$ is full rank, there exists a unique vector \mathbf{c} such that $\mathbf{x}_i = \mathbf{c}\mathbf{A}$. Solve these equations to find \mathbf{c} . L_{x_i} are the indices corresponding to nonzero values in \mathbf{c} . ■

Proposition 3: Let L_{x_i} be the smallest subset of U_x , $|L_{x_i}| = s$, such that $\text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, U_y)) = \text{rank}(\mathbf{T}(L_{x_i}, U_y)) = s$. Then for each $x_j \in L_{x_i}$, $\text{rank}(\mathbf{T}(\{L_{x_i} - x_j, x_i\}, U_y)) = s$.

Proof: Consider the vectors $\mathbf{x}_j \triangleq \mathbf{T}(x_j, U_y)$. From minimality of L_{x_i} , $\mathbf{x}_i = \sum_{j \in L_{x_i}} \alpha_j \mathbf{x}_j$ with $\alpha_j \neq 0$, otherwise, we could have found a smaller set to replace L_{x_i} . Thus, for any $x_k \in L_{x_i}$,

$$\mathbf{x}_i = \beta_k \mathbf{x}_k + \sum_{\substack{x_j \in L_{x_i}, \\ j \neq k}} \beta_j \mathbf{x}_j$$

for some nonzero coefficients β 's over the finite field. Since the vectors $\{\mathbf{x}_j\}$ with $x_j \in L_{x_i}$ are LI, and since given \mathbf{x}_i and all other \mathbf{x}_j apart \mathbf{x}_k we can still retrieve \mathbf{x}_k , the matrix $\mathbf{T}(\{L_{x_i} - x_j, x_i\}, U_y)$ has full rank. ■

C. Examples

Example 1: Exploring an input and the L_x -function. Consider the layer cut in the left Fig. 4 and assume that, during iterations 1 and 2, we have identified the two LI paths \mathcal{P}_1

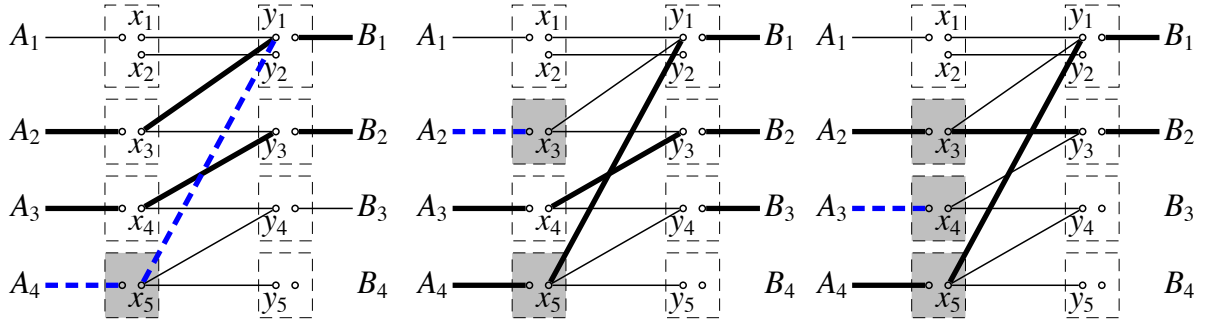


Fig. 4. Assume that bold depict edges in paths \mathcal{P}_1 , \mathcal{P}_2 identified through previous iterations. At iteration 3 a partial path \mathcal{P}_3 arrives at node A_4 , and we explore the edge (x_5, y_1) . We perform rewiring using the L_x function. Left: marked nodes and paths before the L_x function. Middle: marked nodes and paths when substituting x_4 with x_5 . Right: marked nodes and paths when substituting x_3 with x_5 .

and \mathcal{P}_2 that use the bold edges in the figure. Thus,

$$U = \{(x_3, y_1), (x_4, y_3)\}, \quad U_x = \{x_3, x_4\}, \quad U_y = \{y_1, y_3\}, \quad \mathbf{T}(U_x, U_y) = \begin{matrix} & y_1 & y_3 \\ x_3 & \begin{pmatrix} 1 & 1 \end{pmatrix} \\ x_4 & \begin{pmatrix} 0 & 1 \end{pmatrix} \end{matrix}.$$

In iteration 3, assume that we reach node A_4 . We mark this node as visited, and examine the channel input x_5 . There are three possible edges we need to explore: $\{(x_5, y_1), (x_5, y_4), (x_5, y_5)\}$.

- We first examine the edge (x_5, y_1) . This is depicted in the left Fig. 4. Since $y_1 \in U_y$, we are at step $(2 - b)$ of the algorithm. We thus consider the matrix

$$\mathbf{T}(\{U_x, x_5\}, U_y) = \begin{matrix} & y_1 & y_3 \\ x_3 & \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \\ x_4 & \\ x_5 & \end{matrix},$$

and find the set $L_{x_5} = \{x_3, x_4\}$.

We can attempt to substitute each of the $x \in L_x$ with x_5 .

- If we substitute x_3 , we mark A_2 and find another matching: $\{(x_5, y_1), (x_4, y_3)\}$. This is depicted in the middle Fig. 4. Since it is the first time we visit node A_2 , and since there is path arriving at it, we will perform the ϕ -function at this node. We will not describe these steps here, see for such a case example 11. We then call $E(G, \mathcal{P}, \mathcal{M}, A_2)$. Assume this function returns F, i.e., fails to find a path to the destination. We restore the original path matching and continue.

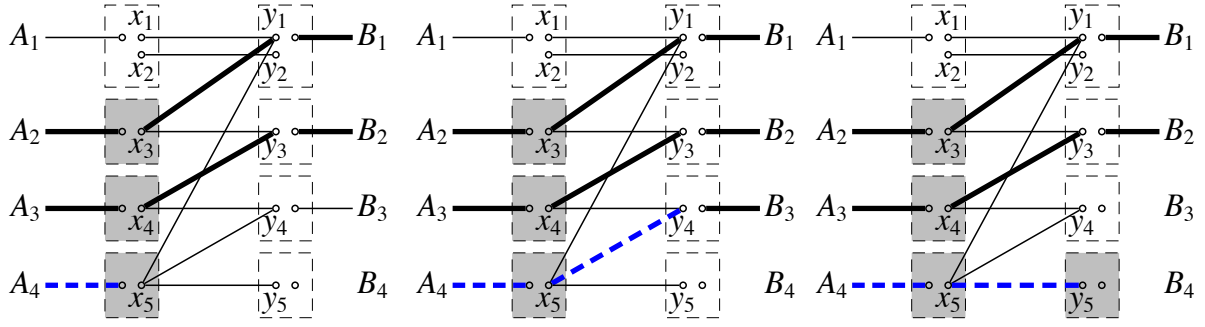


Fig. 5. Continuing from the example in Fig. 4. Failing to use the edge (x_5, y_1) , we will next explore the edges (x_5, y_4) , and (x_5, y_5) . Left: marked nodes and paths. Middle: marked nodes and paths when exploring the edge (x_5, y_4) . Right: marked nodes and paths when exploring the edge (x_5, y_5) .

- If we substitute x_4 : We mark A_3 and find another matching: $\{(x_5, y_1), (x_3, y_3)\}$. This is depicted in the right Fig. 4. We again perform the ϕ -function at node A_3 , bit described in this example. We then call $E(G, \mathcal{P}, \mathcal{M}, A_3)$. Again assume it fails to find a path to the destination. We restore the original path matching and continue.
- We proceed with (x_5, y_4) , as depicted in the middle Fig. 5. Since $y_4 \notin U_Y$, we examine the rank of the matrix

$$\mathbf{T}(\{U_x, x_5\}, \{U_y, y_4\}) = \begin{array}{c} \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{ccc} y_1 & y_3 & y_4 \\ \left(\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{array} \right) \end{array}.$$

Because $\text{rank}(\mathbf{T}(\{U_x, x_5\}, \{U_y, y_4\})) = \text{rank}(\mathbf{T}(U_x, U_y)) = 2$ we are at step $(2 - a - i)$ of the algorithm, and we do not need to take any actions.

- Finally, for the edge (x_5, y_5) , with $y_5 \notin U_Y$, we examine the rank of the matrix

$$\mathbf{T}(\{U_x, x_5\}, \{U_y, y_5\}) = \begin{array}{c} \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{ccc} y_1 & y_3 & y_5 \\ \left(\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array} \right) \end{array}$$

Since $\text{rank}(\mathbf{T}(\{U_x, x_5\}, \{U_y, y_5\})) = \text{rank}(\mathbf{T}(U_x, U_y)) + 1 = 3$, we are at step $(2 - a - ii)$ of the algorithm, and we can use the edge (x_5, y_5) in the path \mathcal{P}_3 . We thus mark node B_4 as visited and continue from there.

That is, we update \mathcal{P} , and we call the function $E(G, \mathcal{P}, \mathcal{M}, B_4)$. Note that since \mathcal{P}_1 and

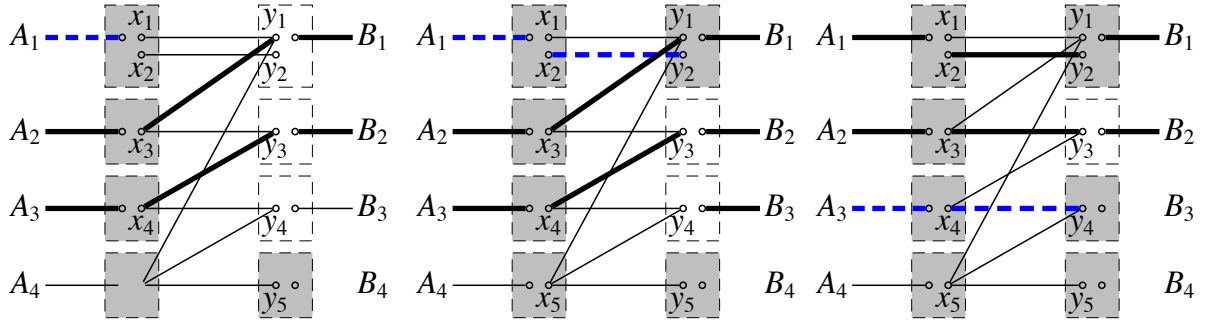


Fig. 6. Continuing from the example in Figs. 4 and 5.

\mathcal{P}_2 do not use node B_4 , we will not perform the ϕ -function at this node. This is depicted in right Fig. 5.

□

We next provide two examples for the ϕ -function.

Example 2: First example for ϕ -function.

Continuing the previous example, assume that we have failed to find a path when exploring A_4 . Suppose that the algorithm continues and suppose that, through some different path, we reach and mark node A_1 , as depicted in the left Fig. 6 (we maintain the marked nodes from the previous algorithm steps during this iteration). We will now explore inputs x_1 and x_2 .

Assume we start by edge (x_2, y_2) . We can use this edge to reach and mark B_1 , as depicted in the middle Fig. 6. Since this is the first time we visit node B_1 , we will perform the ϕ -function.

$$\mathbf{T}(U_x \cup \{x_2\}, U_y \cup \{y_2\} - \{y_1\}) = \mathbf{T}(\{x_2, x_3, x_4\}, \{y_2, y_3\}) = \begin{matrix} & y_1 & y_3 \\ x_3 & \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \\ x_4 & \\ x_5 & \end{matrix}.$$

Consider the transfer matrix where we remove the output y_1 , and use the inputs $\{x_2, x_3, x_4\}$ and the outputs $\{y_2, y_3\}$. Both submatrices $\mathbf{T}(\{x_2, x_3\}, \{y_2, y_3\})$ and $\mathbf{T}(\{x_2, x_4\}, \{y_2, y_3\})$ are full rank, and thus we can explore inputs x_4 and x_3 respectively. We will here describe the steps when selecting the submatrix $\mathbf{T}(\{x_2, x_3\}, \{y_2, y_3\})$.

We find a matching for $\mathbf{T}(\{x_2, x_3\}, \{y_2, y_3\})$, as depicted in the right Fig. 6, and proceed to examine input x_4 . Note that since node $A_3 = A(x_4)$ is already marked, we do not need to

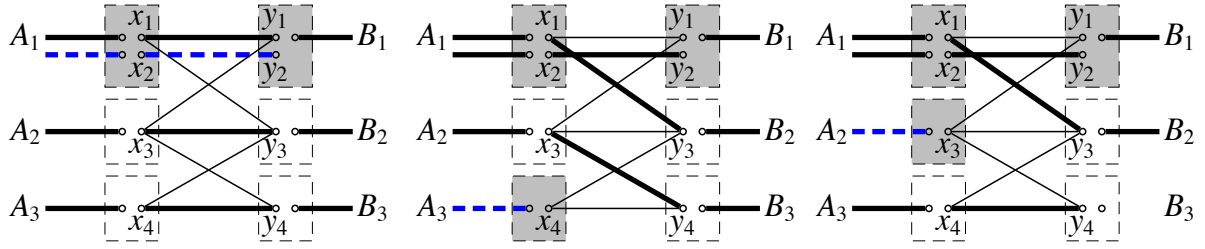


Fig. 7.

explore it again. We observe that we can use the edge (x_4, y_4) , and thus we mark node B_3 and we can further proceed from there. \square

Example 3: Second example for ϕ -function. Consider the layer cut in Fig. 7. Assume during the first three iterations we have identified the paths depicted with bold edges, that is,

$$U = \{(x_1, y_1), (x_3, y_3), (x_4, y_4)\}, \quad U_x = \{x_1, x_3, x_4\}, \quad U_y = \{y_1, y_3, y_4\}, \quad \mathbf{T}(U_x, U_y) = \begin{matrix} & y_1 & y_2 & y_3 \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{matrix}.$$

During iteration 4, we attempt to use edge (x_2, y_2) . Since node B_1 has not been used before, we perform the ϕ -function. We thus consider the matrix

$$\mathbf{T}(\{x_1, x_2, x_3, x_4\}, \{y_2, y_3, y_4\}) = \begin{matrix} & y_2 & y_3 & y_4 \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix} & \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{matrix}$$

which has the full rank submatrices $\mathbf{T}(\{x_1, x_2, x_3\}, \{y_2, y_3, y_4\})$ and $\mathbf{T}(\{x_1, x_2, x_4\}, \{y_2, y_3, y_4\})$. Using the $\mathbf{T}(\{x_1, x_2, x_3\}, \{y_2, y_3, y_4\})$ and the matching depicted in the middle Fig. 7, we can visit node A_3 and explore input x_4 . Note that since A_3 has not been visited before, we need perform the ϕ -function on the node A_3 itself.

If instead we start by utilizing the submatrix $\mathbf{T}(\{x_1, x_2, x_4\}, \{y_2, y_3, y_4\})$ and the matching depicted in the right Fig. 7, we visit node A_2 . Again, since A_2 has not been visited before, we need perform the ϕ -function on the node A_2 as well. \square

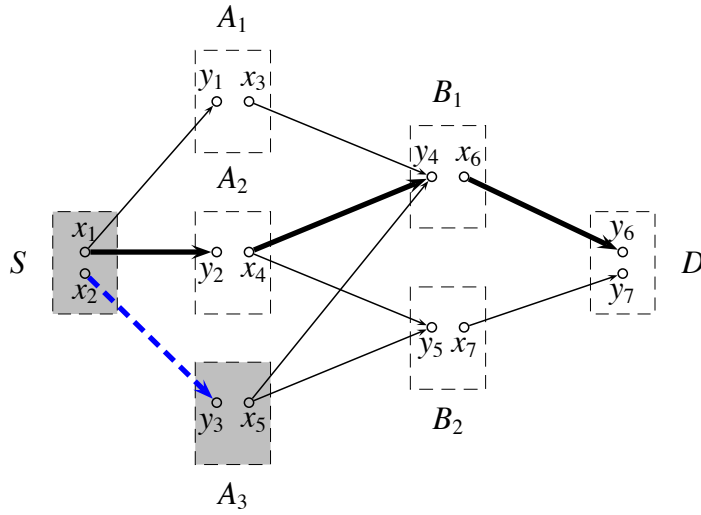


Fig. 8. Path \mathcal{P}_1 identified during the first iteration is depicted in bold. During the second iteration, path \mathcal{P}_2 reached node A_3 .

The next example illustrates how the algorithm runs and performs rewirings across several layers.

Example 4: Example of rewiring across layers. Consider the network depicted in Fig. 8 and assume that the first iteration identified the path $\mathcal{P}_1 = \{(x_2, y_2), (x_3, y_6), x_5, y_9)\}$. During the second iteration, path \mathcal{P}_2 reaches and marks node A_3 , as depicted in Fig. 8. Assume that the algorithm then explores the edge (x_5, y_4) and performs the L_x function. In this case we have that

$$\mathbf{T}(\{x_4, x_5\}, \{y_4\}) = \begin{matrix} & y_4 \\ x_4 & \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ x_5 & \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{matrix}, \quad \text{and } L_{x_5} = \{x_4\}.$$

We thus visit node $A_2 = A(x_4)$. Since it is the first time we visit this node, we perform the ϕ -function at node A_2 . That is, at the first layer, where we now have

$$U = \{(x_1, y_2), (x_2, y_3)\}, \quad U_x = \{x_1, x_2\}, U_y = \{y_2, y_3\}, \quad \mathbf{T}(U_x, U_y) = \begin{matrix} & y_2 & y_3 \\ x_1 & \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ x_2 & \end{matrix}$$

we no longer need to use the output y_2 , and thus can explore inputs x_1 and x_2 . From x_2 we cannot proceed. From x_1 we can use the edge (x_1, y_1) and reach node A_1 as depicted in Fig. 9. We do not need perform the ϕ function at A_1 as there is no additional path using this

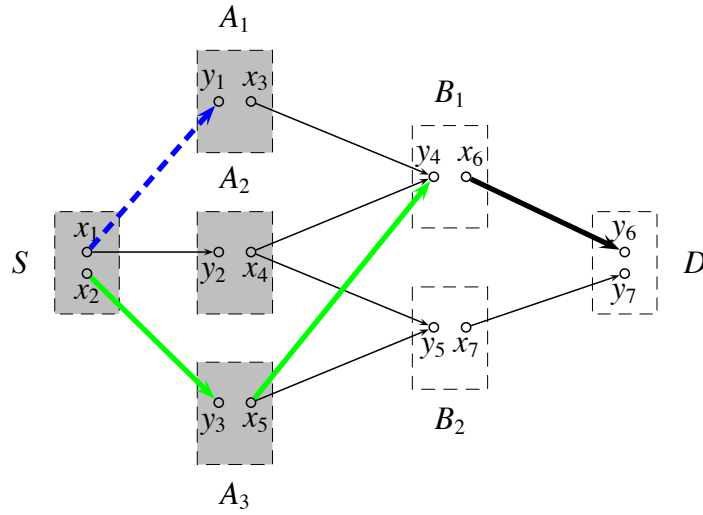


Fig. 9. Continuing from Fig. 8. Resulting configuration after performing the L_x function for edge (x_5, y_4) and the ϕ -function at node A_2 . The potential path \mathcal{P}_2 now reaches node A_1 .

node. We proceed to explore the edge (x_3, y_4) and perform the L_x function for x_3 .

Given that

$$\mathbf{T}(\{x_3, x_5\}, \{y_4\}) = \begin{matrix} & y_4 \\ x_3 & \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ x_5 & \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{matrix}, \quad \text{and } L_{x_3} = \{x_3\},$$

we proceed to re-examine x_5 . Because

$$\mathbf{T}(\{x_3, x_5\}, \{y_4, y_5\}) = \begin{matrix} & y_4 & y_5 \\ x_3 & \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ x_5 & \begin{pmatrix} 1 & 1 \end{pmatrix} \end{matrix}$$

we can now use this edge and proceed to node B_2 . From node B_2 we can use edge (x_7, y_7) to reach the destination and complete path \mathcal{P}_2 .

Note that this is the second time during this iteration that we examine edge (x_5, y_5) . The first time we could not use this edge, due to LD with the used edge (x_4, y_4) . However, after the rewiring, the used edge in this layer became instead (x_3, y_4) , which is LI from (x_5, y_5) .

□

Example 5: Operations over a non-binary field. Consider the network depicted in Fig. 11, which is similar to the network in Fig. 2, only now there is a fixed coefficient associated with each edge over \mathbf{F}_4 . We assume that all these coefficients equal 1, apart from the coefficient

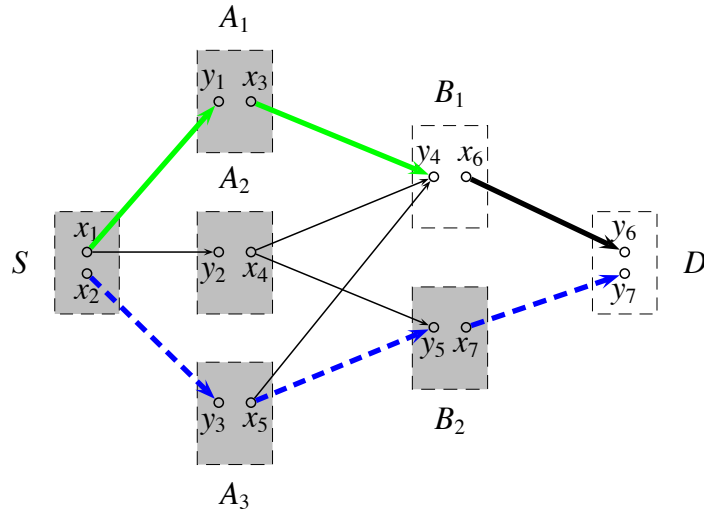


Fig. 10. Continuing from Fig. 9. Resulting configuration after performing the L_x function for edge (x_3, y_4) , and continuing \mathcal{P}_2 from node A_3 to node B_2 and D .

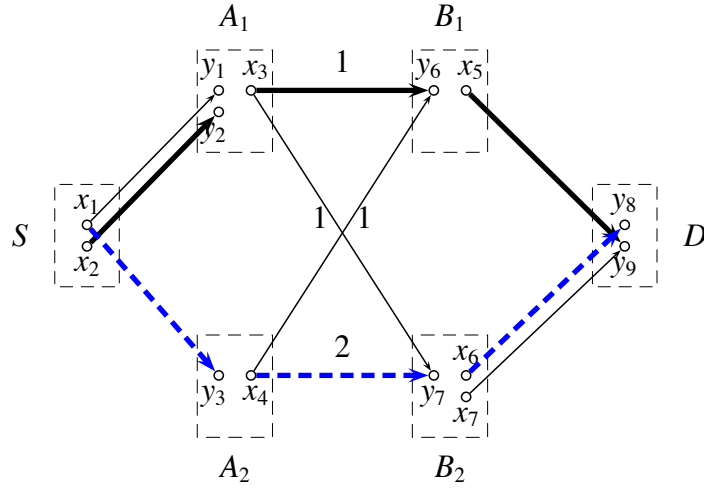


Fig. 11. An example of a nonbinary linear deterministic network. Each edge is associated with a coefficient over \mathbf{F}_4 . All these coefficients equal 1, apart from the coefficient associated with the edge (x_4, y_7) that equals 2.

associated with the edge (x_4, y_7) that equals 2. Operations are now over the field \mathbf{F}_4 . For example, $y_7 = 2x_4 + x_3$.

Assume that the first iteration identified the path $\mathcal{P}_1 = \{(x_2, y_2), (x_3, y_6), (x_5, y_9)\}$. During the second iteration, assume that we use at the first layer the edge (x_1, y_3) , and arrive at layer 2. At this layer, $U_x = \{x_3\}$ and $U_y = \{y_6\}$. To use edge (x_4, y_7) , we examine whether the matrix

$$\mathbf{T}(\{x_3, x_4\}, \{y_6, y_7\}) = \begin{matrix} & y_6 & y_7 \\ x_3 & \begin{pmatrix} 1 & 1 \end{pmatrix} \\ x_4 & \begin{pmatrix} 1 & 2 \end{pmatrix} \end{matrix}$$

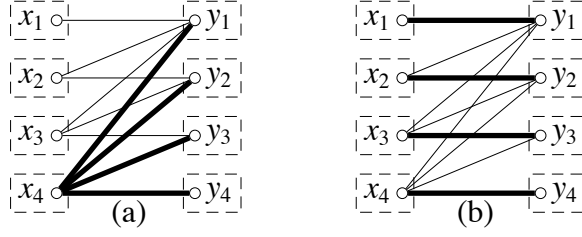


Fig. 12. A layer-cut and (a) the traditional approach where interference is treated as noise, (b) the approach where interference is allowed.

is full rank over \mathbf{F}_4 . As indeed it is, we can reach node B_2 , and from there using edge (x_6, y_8) complete \mathcal{P}_2 . Note that in the binary example in Fig. 2, we could only identify one path. \square

We conclude with an example that shows the benefits of not treating interference as noise.

Example 6: Benefits from constructive use of interference. The traditional approach adopted today in wireless networks is that if one or more transmitted signals interfere with a received signal, they are treated as noise. Such interference is avoided through scheduling. This approach can lead to significant loss of capacity. Consider a network that has the layer-cut depicted in Fig. 12. Fig. 12(a) depicts the traditional solution: treating interference as noise implies that we cannot simultaneously have two broadcast transmissions that interfere, and thus we can have at most one broadcast transmission. Fig. 12(b) shows that, if interference is allowed, we can in fact use four LI edges through this cut (the example is easily generalized to N nodes leading to $O(N)$ benefits). Indeed, the transfer matrix associated with this cut,

$$\mathbf{T}(\{x_1, x_2, x_3, x_4\}, \{y_1, y_2, y_3, y_4\}) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

has rank four. This matrix coincides with the transformation matrix of the highlighted edges. \square

D. Main Result

Our main result is the following theorem.

Theorem 1: The unicast algorithm identifies C LI paths, where C is the min-cut value between the source-destination pair in a linear deterministic network.

In particular, the number of the paths identified by the algorithm equals the rank of the

transfer matrix between the inputs in V_1 and the outputs in V_2 , where V_1 are the marked and V_2 are the unmarked nodes when the algorithm stops.

Proof: Based on the algorithm, it is clear that when the algorithm stops, the provided output is a set of linearly independent source-destination paths \mathcal{P} .

Let K denote the number of these paths; this implies that the algorithm stops, i.e., fails to find an additional path, during iteration $K + 1$. Since K can never exceed the rank of a source-destination cut, i.e., $K \leq C$, it suffices to find a cut whose capacity is not bigger than the number of paths identified by our algorithm. Let V_1 be the set of all marked (visited) vertices and V_2 be the other vertices during iteration $K + 1$, when the algorithm stops. Clearly, (V_1, V_2) is a source destination cut.

Consider now the matrix $\mathbf{T}(V_1, V_2)$, where, by a slight abuse of notation, the set of rows of this matrix correspond to the inputs x in V_1 and the set of columns to the outputs y in nodes in V_2 respectively. By appropriate ordering of these inputs and outputs we can bring the transfer matrix in to a block diagonal form, in which every block corresponds to a layer of the network. More precisely, if W_i (W'_i) is the set of visited (unvisited) nodes in the i -th layer then $\mathbf{T}(V_1, V_2)$ can be regarded as a block diagonal matrix whose i -th block is $\mathbf{T}(W_i, W'_{i+1})$. For clarity we have collected all the notation we use in this proof in Table III-D.

We will show in Lemma 1 that for every integer $1 \leq i \leq \Lambda$ it holds that,

$$\begin{aligned} \text{rank}(\mathbf{T}(W_i, W'_{i+1})) &\leq \\ &|\{e = vv' \in E(G) | v \in W_i, v' \in W'_{i+1}, e \in U\}| - |\{e = vv' \in E(G) | v \in W'_i, v' \in W_{i+1}, e \in U\}| \end{aligned} \quad (1)$$

where recall that we denote by U the set of used edges by paths in \mathcal{P} at layer i . That is, if $e \in U$, then it belongs in some path j , i.e., $e \in \mathcal{P}_j$ (and more generally e belongs in the set of all used edges in the graph, i.e., $e \in \mathcal{P}$). Also, from the structure of the layer network, the total number of paths equals the number of used edges in each layer, namely, $|U| = K$. Now

$$\begin{aligned} &\text{rank}(\mathbf{T}(V_1, V_2)) \\ &\stackrel{(a)}{=} \sum_{i=1}^{\Lambda} \text{rank}(\mathbf{T}(W_i, W'_{i+1})) \\ &\stackrel{(b)}{\leq} |\{e = vv' | v \text{ is visited but } v' \text{ is not visited, } e \in \mathcal{P}\}| - |\{e = vv' | v \text{ is not visited but } \\ &v' \text{ is visited, } e \in \mathcal{P}\}| \\ &\stackrel{(c)}{=} K. \end{aligned} \quad (2)$$

Equality (a) holds from the fact that the rank of any block diagonal matrix is the sum of the rank of its blocks. Inequality (b) follows directly from Lemma 1 that will prove in the following. Finally, equality (c) holds because for each source-destination path \mathcal{P}_i the “used” edges by \mathcal{P}_i contribute exactly one in the difference, that is,

$$\begin{aligned} & |\{e = vv' | v \text{ is visited but } v' \text{ is not visited, } e \in \mathcal{P}_i\}| - \\ & - |\{e = vv' | v \text{ is not visited but } v' \text{ is visited, } e \in \mathcal{P}_i\}| = 1 \end{aligned} \quad (3)$$

Indeed, given a cut (V_1, V_2) , with $S \in V_1$ and $D \in V_2$, for \mathcal{P}_i to connect S to D , it must cross at least one time from V_1 to V_2 . If it crosses $m \geq 1$ times from V_1 to V_2 , then it also has to cross $m - 1$ times from V_2 to V_1 . ■

Lemma 1: For every integer $1 \leq i \leq \Lambda$ it holds that,

$$\begin{aligned} & \text{rank}(\mathbf{T}(W_i, W'_{i+1})) \leq \\ & |\{e = vv' \in E(G) | v \in W_i, v' \in W'_{i+1}, e \in U\}| - |\{e = vv' \in E(G) | v \in W'_i, v' \in W_{i+1}, e \in U\}| \end{aligned} \quad (4)$$

Proof: Fix an integer $1 \leq i \leq \Lambda$. Recall that we denote by U the set of used channels in this layer (dropping the index i for simplicity), U_x their inputs and U_y their outputs. Additionally, let U_{B_x} be the set of all the inputs of the nodes in W_i and U'_{B_x} be the set of all the visited inputs in the current layer which appear in some identified path. That is, $U'_{B_x} = U_{B_x} \cap U_x$. Let U_{B_y} be the set of all the outputs that are in the $i + 1$ -st layer and are not visited and U'_{B_y} be those outputs of U_{B_y} which appear on some identified path (i.e., used outputs). That is, $U'_{B_y} = U_{B_y} \cap U_y$. The notation is summarized in Table III-D.

We are interested in calculating the rank of the matrix $\mathbf{T}(W_i, W'_{i+1}) = \mathbf{T}(U_{B_x}, U_{B_y})$. Note that we can split the columns of $\mathbf{T}(U_{B_x}, U_{B_y})$ into two parts, one corresponding to the used and unmarked outputs, i.e., U'_{B_y} , and the other corresponding to the unused and unmarked outputs, $U_{B_y} - U'_{B_y}$. Similarly, we can split the rows again into two parts, one corresponding to the used and marked inputs, U'_{B_x} , and the other to the unused and marked inputs, $U_{B_x} - U'_{B_x}$.

Our proof proceeds as follows. Lemmas 3 and 4 prove that all the rows of $\mathbf{T}(U_{B_x}, U_{B_y})$ that belong to the second part (in $U_{B_x} - U'_{B_x}$) are in the span of the rows corresponding to the inputs in the first part (in U'_{B_x}). As a result,

$$\text{rank} \mathbf{T}(U_{B_x}, U_{B_y}) = \text{rank} \mathbf{T}(U'_{B_x}, U_{B_y}). \quad (5)$$

Lemma 5 builds on this result to prove that

$$\text{rankT}(U_{B_x}, U_{B_y}) = |U'_{B_x}| - (|U_y| - |U'_{B_y}|). \quad (6)$$

Showing that (6) holds is the main technical part of this proof. Now we distinguish three cases for each edge $e = (x, y) \in U$:

- 1) $x \in U'_{B_x}$ and $y \notin \{U_y - U'_{B_y}\}$: the edge contributes value “one” only in $|U'_{B_x}|$,
- 2) $x \notin U'_{B_x}$ and $y \in \{U_y - U'_{B_y}\}$: the edge contributes value “one” only in $(|U_y| - |U'_{B_y}|)$
- 3) $x \in U'_{B_x}$ and $y \in \{U_y - U'_{B_y}\}$: then the edge contributes value “one” both in $|U'_{B_x}|$ and in $(|U_y| - |U'_{B_y}|)$ and thus does not affect the quantity $|U'_{B_x}| - (|U_y| - |U'_{B_y}|)$.

Thus

$$\begin{aligned} & |U'_{B_x}| - (|U_y| - |U'_{B_y}|) = \\ & |\{e = vv' \in E(G) | v \in U'_{B_x}, v' \in U'_{B_y}, e \in U\}| - |\{e = vv' \in E(G) | v \notin U'_{B_x}, v' \in U_{B_y}, e \in U\}| \\ & = |\{e = vv' \in E(G) | v \in W_i, v' \in W'_{i+1}, e \in U\}| - |\{e = vv' \in E(G) | v \in W'_i, v' \in W_{i+1}, e \in U\}| \end{aligned}$$

and our proof is concluded. ■

Before we continue, we need to introduce some additional notation.

When iteration $K + 1$ starts, at the layer we are examining, we have identified from the previous iterations a set of used edges U , with corresponding set of inputs and outputs U_x and U_y respectively. As the algorithm attempts to find the $K + 1$ path, it may perform some rewirings inside this layer (due to consecutive executions for example of several L_x and ϕ -functions). Thus, when input x_i gets marked and starts to be explored by the algorithm, this input might perceive as used a different set of edges than U . We will denote by $\mathcal{R}^{(i)}$ the set of edges that input x_i perceives as used (by the K paths), and $\mathcal{R}_x^{(i)}, \mathcal{R}_y^{(i)}$ the corresponding sets of used inputs and outputs. Note that, while all the edges emanating from x_i are examined, for all of them the algorithm will assume the same set of used edges $\mathcal{R}^{(i)}$.

Now, from assumption, the iteration $K + 1$ fails to find a path from S to D . Thus, although several rewirings might be attempted, because iteration $K + 1$ fails, when the algorithm stops we have reverted to the original set U .

Lemma 2: For all $x_i \in U_{B_x} - U'_{B_x}$ it holds that

$$\text{rankT}(\{\mathcal{R}_x^{(i)}, x_i\}, \{\mathcal{R}_y^{(i)}, U_{B_y} - U'_{B_y}\}) = \text{rankT}(\mathcal{R}_x^{(i)}, \{\mathcal{R}_y^{(i)}, U_{B_y} - U'_{B_y}\})$$

In particular, there exists a minimal set of rows $L_{x_i} \subseteq \mathcal{R}_x^{(i)}$ such that

$$\text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, \{\mathcal{R}_y^{(i)}, U_{By} - U'_{By}\})) = \text{rank}(\mathbf{T}(L_{x_i}, \{\mathcal{R}_y^{(i)}, U_{By} - U'_{By}\})). \quad (7)$$

Proof: For this proof only we also use the following notation. Assume that $\text{rank}(\mathbf{T}(\{\mathcal{R}^{(i)}, x_i\}, Z)) = \text{rank}(\mathbf{T}(\mathcal{R}^{(i)}, Z))$ for some set of columns Z . Define $L_{x_i}(Z)$ to be the smallest subset of $\mathcal{R}^{(i)}$ that contains x_i in its span, i.e.,

$$\text{rank}(\mathbf{T}(\{L_{x_i}(Z), x_i\}, Z)) = \text{rank}(\mathbf{T}(L_{x_i}(Z), Z)).$$

We will use for abbreviation $L_{x_i} = L_{x_i}(\mathcal{R}_y^{(i)})$.

Decompose the column indices of the matrix $\mathbf{T}(\mathcal{R}_x^{(i)}, \{\mathcal{R}_y^{(i)}, U_{By} - U'_{By}\})$ in the following 4 nonoverlapping parts: $[\mathcal{R}_y^{(i)}, W_1, W_L, W_0]$. Here

- $\mathcal{R}_y^{(i)}$ contains all the used y 's,
- W_1 contains all y_j such that the edges (x_i, y_j) exist but cannot be used due to LD,
- W_L contains all the remaining $y_k \in W_{By}$ that have at least one nonzero value in each column (i.e., the set of all y columns where at least one edge (x_k, y_k) with $x_k \in L_{x_i}(W_{By})$ exists, but x_i has zero value), and
- W_0 contains all zero columns (this is the set of y 's associated with x 's not in the set $\{L_{x_i}(W_{By}), x_i\}$).

We underline that the set of columns $U_{By} - U'_{By} = \{W_1, W_L, W_0\}$ is the set of unmarked unused outputs *at the end* of the iteration $K + 1$, and is the same independently of the set of outputs in $\mathcal{R}_y^{(i)}$. Note that, because $\mathcal{R}_y^{(i)}$ can contain either outputs that belong in U_y (that thus are used) and/or outputs obtained through the execution of the ϕ -function (and thus are marked), has by definition zero overlap with the set $U_{By} - U'_{By}$ which contains outputs that are both unmarked and not used.

To prove the lemma, it is sufficient to prove that the following equation holds.

$$\begin{aligned} L_{x_i} &= L_{x_i}(\mathcal{R}_y^{(i)}) \stackrel{(a)}{=} L_{x_i}(\{\mathcal{R}^{(i)}, W_1\}) \\ &\stackrel{(b)}{=} L_{x_i}(\{\mathcal{R}^{(i)}, W_L\}) \stackrel{(c)}{=} L_{x_i}(\{\mathcal{R}^{(i)}, W_0\}). \end{aligned} \quad (8)$$

(a) : To prove (a) we need to show that $L_{x_i}(\mathcal{R}_y^{(i)}) = L_{x_i}(\mathcal{R}_y^{(i)}, W_1)$, that is,

$$\text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, \{\mathcal{R}_y^{(i)}, W_1\})) = \text{rank}(\mathbf{T}(L_{x_i}, \{\mathcal{R}_y^{(i)}, W_1\})).$$

Since the matrix $\mathbf{T}(\mathcal{R}_x^{(i)}, \mathcal{R}_y^{(i)})$ is full rank, the row $\mathbf{T}(x_i, \mathcal{R}_y^{(i)})$ belongs in the span of this matrix, and thus there exist nonzero coefficients $\{\alpha_j\}$ in \mathbf{F}_q such that

$$\mathbf{T}(x_i, \mathcal{R}_y^{(i)}) = \sum_{x_j \in L_{x_i}(\mathcal{R}_y^{(i)})} \alpha_j \mathbf{T}(x_j, \mathcal{R}_y^{(i)}). \quad (9)$$

Note that for each $y_j \in W_1$, there also exist nonzero coefficients $\{\beta_j\}$ in \mathbf{F}_q such that

$$\text{rank}(\mathbf{T}(\{\mathcal{R}_x^{(i)}, x_i\}, \{\mathcal{R}_y^{(i)}, y_j\})) = \text{rank}(\mathbf{T}(\mathcal{R}_x^{(i)}, \{\mathcal{R}_y^{(i)}, y_j\})) \quad (10)$$

otherwise the node $A(y_j)$ would have been visited and marked and $y_j \notin W_1$. Thus $\mathbf{T}(x_i, \{\mathcal{R}_y^{(i)}, W_1\})$ belongs in the span of $\mathbf{T}(\mathcal{R}_x^{(i)}, \{\mathcal{R}_y^{(i)}, W_1\})$, and

$$\mathbf{T}(x_i, \{\mathcal{R}_y^{(i)}, W_1\}) = \sum_{x_j \in L_{x_i}(\mathcal{R}_y^{(i)}, W_1)} \beta_j \mathbf{T}(x_j, \{\mathcal{R}_y^{(i)}, W_1\}). \quad (11)$$

Expurgating from both sides of (11) the columns of W_1 results in an equation that still holds for the expurgated vectors and has only columns corresponding to $\mathcal{R}_y^{(i)}$. From LI of all vectors $\mathbf{T}(x, \mathcal{R}_y^{(i)})$, $x \in \mathcal{R}_x^{(i)}$, none of these expurgated vectors is identically zero. Moreover, from minimality of $L_{x_i}(\mathcal{R}_y^{(i)})$ the expansion (9) is unique. We thus conclude that $\alpha_j = \beta_j$ and $L_{x_i} = L_{x_i}(\mathcal{R}_y^{(i)}) = L_{x_i}(\mathcal{R}_y^{(i)}, W_1)$.

(b) : We will now argue that $L_{x_i}(\mathcal{R}_y^{(i)}) = L_{x_i}(\{\mathcal{R}_y^{(i)}, W_L\})$. Consider a specific $y_k \in W_L$ that has a nonzero value in a row $x_k \in L_{x_i}$. That is, there exists an edge (x_k, y_k) with $x_k \in L_{x_i}$ and $y_k \in W_L$.

During the algorithm, we will at some point “release” x_k from the set of used edges and replace it with x_i . We will then attempt to explore x_k , assuming the set of used edges $\mathcal{R}^{(i)}$. Note that x_k might have already been explored before using a different set of used edges $\mathcal{R}^{(k)}$. However, our algorithm will for each x_i explore all inputs in the set L_{x_i} using $\mathcal{R}^{(i)}$ again, even though these might have been explored before.

If the matrix $\mathbf{T}(\{\mathcal{R}_x^{(i)}, x_i\}, \{\mathcal{R}_y^{(i)}, y_k\})$ is full rank, then the node $A(y_k)$ will be visited and $y_k \notin U_{By}$ which is a contradiction. Thus the matrix $\mathbf{T}(\{\mathcal{R}_x^{(i)}, x_i\}, \{\mathcal{R}_y^{(i)}, y_k\})$ is not full rank. Consider then the set $L_{x_i}(\{\mathcal{R}_y^{(i)}, y_k\})$. Applying a similar argument as in (a), we have that

$$\mathbf{T}(x_i, \mathcal{R}_y^{(i)}) = \sum_{x_j \in L_{x_i}(\mathcal{R}_y^{(i)})} \alpha_j \mathbf{T}(x_j, \mathcal{R}_y^{(i)}) \quad (12)$$

and

$$\mathbf{T}(x_i, \{\mathcal{R}_y^{(i)}, y_k\}) = \sum_{x_j \in L_{x_k}(\mathcal{R}_y^{(i)}, y_k)} \beta_j \mathbf{T}(x_k, \{\mathcal{R}_y^{(i)}, y_k\}). \quad (13)$$

Expurgating the column corresponding to y_k we conclude that $L_{x_i}(\mathcal{R}_y^{(i)} \cup \{y_k\}) = L_{x_i}(\mathcal{R}_y^{(i)})$. Repeating for all $y_k \in W_L$ concludes (b).

(c) : Clearly it also holds that

$$\text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, \mathcal{R}_y^{(i)})) = \text{rank}(\mathbf{T}(L_{x_i}, \{\mathcal{R}_y^{(i)}, W_0\})),$$

which concludes the proof of this lemma. ■

Lemma 3: For each $x_i \in U_{B_x} - U'_{B_x}$, the vector $\mathbf{T}(x_i, \{\mathcal{U}_y^{(i)}, U_{B_y} - U'_{B_y}\})$ belongs in the span of the matrix $\mathbf{T}(U'_{B_x}, \{\mathcal{U}_y^{(i)}, U_{B_y} - U'_{B_y}\})$, where $\mathcal{U}_y^{(i)}$ denotes the set of unmarked outputs in the set U_y .

Proof: Note that all unmarked outputs in U_y are included in $\mathcal{R}_y^{(i)}$, and thus, $\mathcal{U}_y^{(i)} \subseteq \mathcal{R}_y^{(i)} \cap U_y$.

Order the inputs $x_i \in U_{B_x} - U'_{B_x}$ according to the order with which they are for the first time visited. That is, x_1 is the first unused input that is explored inside layer i and during iteration $K + 1$, x_2 the second one, etc. We will prove our claim through induction.

Induction Step 1: When x_1 , the first input, gets visited, clearly $\mathcal{R}^{(1)} = U$, and $\mathcal{U}_y^{(1)} = U_y$ since to perform a rewiring using a new output, we need to have already explored at least one input. From lemma 2 we know that the vector $\mathbf{T}(x_1, \{U_y, U_{B_y} - U'_{B_y}\})$ belongs in the span of the matrix $\mathbf{T}(U_{B_x}, \{U_y, U_{B_y} - U'_{B_y}\})$ and in particular from (7) belongs in the span of the matrix $\mathbf{T}(L_{x_1}, \{U_y, U_{B_y} - U'_{B_y}\})$.

It is then sufficient to prove that the inputs in L_{x_1} belong in marked nodes, i.e., $L_{x_1} \subseteq U'_{B_x}$. But this holds, because of the algorithm steps when we visit x_1 . In particular, when x_1 is explored, all nodes with $x \in L_{x_1}$ are visited, marked, and explored assuming the set of used edges U . Thus $L_{x_1} \subseteq U'_{B_x}$, and

$$\text{rank}(\mathbf{T}(\{U'_{B_x}, x_1\}, \{U_y, U_{B_y} - U'_{B_y}\})) = \text{rank}(\mathbf{T}(U'_{B_x}, \{U_y, U_{B_y} - U'_{B_y}\})) \quad (14)$$

Induction Step k: Assume that for $1 \leq i \leq k$ $\mathbf{T}(x_i, \{\mathcal{U}_y^{(i)}, U_{B_y} - U'_{B_y}\})$ belongs in the span of the matrix $\mathbf{T}(U'_{B_x}, \{\mathcal{U}_y^{(i)}, U_{B_y} - U'_{B_y}\})$.

Induction Step k+1: From lemma 2, we know that the vector $\mathbf{T}(x_{k+1}, \{\mathcal{R}_y^{(k+1)}, U_{B_y} - U'_{B_y}\})$ belongs in the span of the matrix $\mathbf{T}(\mathcal{R}_x^{(k+1)}, \{\mathcal{R}_y^{(k+1)}, U_{B_y} - U'_{B_y}\})$, and in particular in the span

of the matrix $\mathbf{T}(L_x^{(k+1)}, \{\mathcal{R}_y^{(k+1)}, U_{By} - U'_{By}\})$, where $L_x^{(k+1)} \subseteq \mathcal{R}_x^{(k+1)}$. Removing the columns that are not in $\mathcal{U}_y^{(i)}$, we get that the row $\mathbf{T}(x_{k+1}, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\})$ is in the span of the rows $\mathbf{T}(L_x^{(k+1)}, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\})$. Now, all $x \in L_x^{(k+1)}$ are visited and marked during the algorithm. For each such x , if $x \in U_x$, then x will appear in U'_{Bx} . If on the other hand $x \in \mathcal{R}_x^{(k+1)}$ but $x \notin U_x$, then x is one of $\{x_1, \dots, x_k\}$ since $\mathcal{R}_x^{(k+1)}$ can only differ from U_x on marked inputs. From induction, for each $x_i \in \{x_1, \dots, x_k\}$ the row vector $\mathbf{T}(x_i, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\})$ belongs in the span of the matrix $\mathbf{T}(U'_{Bx}, \{\mathcal{U}_y^{(i)} \cap U_y, U_{By} - U'_{By}\})$. Moreover, $\mathcal{U}_y^{(k+1)} \subseteq \mathcal{U}_y^{(i)}$, $i < k + 1$, since, if some outputs are unmarked during iteration $K + 1$, they also are unmarked during the previous iterations. This concludes this proof. \blacksquare

Lemma 4: In the matrix $\mathbf{T}(U_{Bx}, U_{By})$ each row corresponding to unused marked inputs, i.e., $x_i \in U_{Bx} - U'_{Bx}$, is in the span of the rows corresponding to inputs in U'_{Bx} , and thus $\text{rank}\mathbf{T}(U_{Bx}, U_{By}) = \text{rank}\mathbf{T}(U'_{Bx}, U_{By})$.

Proof: From Lemma 3, for each $x_i \in U_{Bx} - U'_{Bx}$, we know that the row vector $\mathbf{T}(x_i, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\})$ belongs in the span of the matrix $\mathbf{T}(U'_{Bx}, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\})$. That is,

$$\mathbf{T}(x_i, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\}) = \sum_{x_j \in U'_{Bx}} \alpha_j \mathbf{T}(x_j, \{\mathcal{U}_y^{(i)}, U_{By} - U'_{By}\}) \quad (15)$$

for some $\alpha_j \in \mathbf{F}_q$. Next, note that U'_{By} is a subset of $\mathcal{U}_y^{(i)}$ for each i . This is because, at each rewiring, $\mathcal{U}_y^{(i)}$ can differ from U_y only on *marked* outputs. But U'_{By} is the set of used and *unmarked* outputs, and thus $U'_{By} \subseteq \mathcal{U}_y^{(i)}$. Removing some columns from both sides of (15) we get that

$$\mathbf{T}(x_i, U_{By}) = \mathbf{T}(x_i, \{U'_{By}, U_{By} - U'_{By}\}) = \sum_{x_j \in U'_{Bx}} \alpha_j \mathbf{T}(x_j, \{U'_{By}, U_{By} - U'_{By}\}) = \sum_{x_j \in U'_{Bx}} \alpha_j \mathbf{T}(x_j, U_{By})$$

and the claim follows. \blacksquare

Lemma 5: The rank of the matrix $\mathbf{T}(U_{Bx}, U_{By})$ can be upper bounded as

$$\text{rank}\mathbf{T}(U_{Bx}, U_{By}) \leq |U'_{Bx}| - (|U_y| - |U'_{By}|).$$

Proof: Consider the matrix $\mathbf{A} \triangleq \mathbf{T}(U'_{Bx}, \{U_y, U_{By} - U'_{By}\})$. This matrix has less rows than $\mathbf{T}(U_{Bx}, U_{By})$ as it does not contain the rows in $U_{Bx} - U'_{Bx}$, and has more columns than $\mathbf{T}(U_{Bx}, U_{By})$ as it contains the additional columns corresponding to the outputs $U_y - U'_{By}$. The idea in this proof is to gradually change matrix \mathbf{A} , by sequentially adding rows and by removing columns, until we create the matrix $\mathbf{T}(U_{Bx}, U_{By})$, taking into account how each

operation affects the rank.

Order the marked outputs in U_y , i.e., the outputs in $U_y - U'_{By}$ that we need remove, according to the time they got marked, i.e., y_1 is the output that got marked first when node $A(y_1)$ is visited, y_2 the one that got marked second, etc. Now, assume that at the time when output y_1 is visited, j_1 unused inputs (not in U_x) have already been visited and marked (note that $j_1 \geq 1$, if $j_1 = 0$ it is not possible to mark y_1). In general, when output y_k is visited, we will have that j_k inputs in $U_{Bx} - U'_{Bx}$ are marked, with $j_1 \leq j_2 \leq j_3 \dots \leq j_L$ and $L \triangleq |U_{Bx}| - |U'_{Bx}|$.

Our starting point is that the matrix \mathbf{A} has rank $|U'_{Bx}|$, i.e., all its rows are linearly independent. Indeed, since the $K \times K$ matrix $\mathbf{T}(U_x, U_y)$ is full rank and $U'_{Bx} \subseteq U_x$, the rows $\mathbf{T}(U'_{Bx}, U_y)$ are LI, and as a result so are the rows $\mathbf{T}(U'_{Bx}, \{U_y, U_{By} - U'_{By}\})$.

We are going to perform $L = |U_y| - |U'_{By}|$ steps, creating a sequence of matrices $\{\mathbf{A}_0 = \mathbf{A}, \mathbf{A}_1, \dots, \mathbf{A}_L\}$ where at step k , $k = 1, \dots, L$, we first add to matrix \mathbf{A}_{k-1} the rows $\{x_{j_{k-1}+1} \dots x_{j_k}\}$ and then we remove the output y_k in $(U_y - U'_{By})$ to create the matrix \mathbf{A}_k .

Step 1: Removing output y_1 .

Let $\mathcal{R}^{(j_1)}$ be the set of perceived used edges when y_1 is marked. Since this is the first time an output in U_y is marked and the ϕ function is executed, $\mathcal{R}_y^{(j)} = U_y$, for all $j \leq j_1$.

We know from lemma 3 that the rows $\mathbf{T}(x_i, \{U_y, U_{By} - U'_{By}\})$, $1 \leq i \leq j_1$ belong in the span of the matrix $\mathbf{T}(U'_{Bx}, \{U_y, U_{By} - U'_{By}\})$. Thus adding these rows to matrix \mathbf{A} does not increase its rank.

From lemma 6, there exist a set of rows $S(y_1)$ with $S(y_1) \subseteq \{\mathcal{R}_x^{(j_1)}, x_1, \dots, x_{j_1}\}$ such that, removing the column y_1 drops the rank of the matrix $\mathbf{T}(S(y_1), \{\mathcal{R}_y^{(j_1)}, U_{By} - U'_{By}\})$ from $|S(y_1)|$ to $|S(y_1)| - 1$. In other words, the column $\mathbf{T}(S(y_1), y_1)$ is LI from all the columns of the matrix $\mathbf{T}(S(y_1), \{\mathcal{R}_y^{(j_1)} - y_1, U_{By} - U'_{By}\})$.

Notice that when the node $A(y_1)$ gets visited during iteration $K + 1$, we will execute the ϕ -function for output y_1 . As a result, all the nodes with inputs in $S(y_1)$ will be visited and marked by the algorithm during iteration $K + 1$. Thus we know that $S(y_1) \subseteq \{U'_{Bx}, x_1, \dots, x_{j_1}\}$, that is, they form part of the set of marked inputs by the algorithm.

Since the ‘‘partial’’ column $\mathbf{T}(S(y_1), y_1)$ is LI from the columns in the matrix $\mathbf{T}(S(y_1), \{U_y - y_1, U_{By} - U'_{By}\})$, it follows immediately that the column $\mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_1}\}, y_1)$ is LI from the columns in the matrix $\mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_1}\}, \{\mathcal{R}_y^{(j_1)} - y_1, U_{By} - U'_{By}\})$. Thus if we drop the column y_1 from the matrix $\mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_1}\}, \{U_y, U_{By} - U'_{By}\})$ the resulting matrix $\mathbf{A}_1 \triangleq \mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_1}\}, \{\mathcal{R}_y^{(j_1)} - y_1, U_{By} - U'_{By}\})$ has rank $|U'_{Bx}| - 1$.

Step k: Removing output y_k .

We start from the matrix $\mathbf{A}_{k-1} \triangleq \mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_{k-1}}\}, \{\mathcal{R}_y^{(j_1)} - y_1 - \dots - y_{k-1}, U_{By} - U'_{By}\})$ that has rank $|U'_{Bx}| - (k - 1)$. From lemma 3 the rows $\mathbf{T}(x_j, \{U_y - y_1 - \dots - y_{k-1}, U_{By} - U'_{By}\})$, $j_{k-1} \leq j \leq j_k$ belong in the span of the matrix $\mathbf{T}(U'_{Bx}, \{U_y - y_1 - \dots - y_{k-1}, U_{By} - U'_{By}\})$. Thus adding these rows to matrix \mathbf{A}_{k-1} does not increase its rank.

On the other hand, from lemma 6 there exists a set of LI rows $S(y_k) \subseteq \{\mathcal{R}_x^{(j_k)}, x_1, \dots, x_{j_{k-1}}\}$ such that removing the column y_k from the matrix $\mathbf{T}(S(y_k), \{\mathcal{R}_y^{(j_k)}, U_{By} - U'_{By}\})$ drops the rank of this matrix from $|S(y_k)|$ to $|S(y_k)| - 1$. In other words, the column $\mathbf{T}(S(y_k), y_k)$ is LI from all the columns of the matrix $\mathbf{T}(S(y_k), \{\mathcal{R}_y^{(j_k)} - y_k, U_{By} - U'_{By}\})$. But $\mathcal{R}_y^{(j_k)}$ contains all the outputs in $U_y - y_1 - \dots - y_{k-1}$, and thus the column $\mathbf{T}(S(y_k), y_k)$ does not belong in the span of the columns $\mathbf{T}(S(y_k), \{U_y - y_1 - \dots - y_{k-1}, U_{By} - U'_{By}\})$.

Similar to before because the ϕ -function will be executed at y_k , all the inputs in $S(y_k)$ are marked and $S(y_k) \subset \{U'_{Bx}, x_1, \dots, x_{j_k}\}$. It again follows immediately that the column $\mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_k}\}, y_k)$ is LI from the columns in the matrix $\mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_k}\}, \{\mathcal{R}_y^{(j_1)} - y_1 - \dots - y_k, U_{By} - U'_{By}\})$. Thus if we drop the column y_k from the matrix $\mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_k}\}, \{U_y - y_1 - \dots - y_{k-1}, U_{By} - U'_{By}\})$ the resulting matrix $\mathbf{A}_k \triangleq \mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_k}\}, \{\mathcal{R}_y^{(j_1)} - y_1 - \dots - y_k, U_{By} - U'_{By}\})$ has rank $|U'_{Bx}| - k$.

Final step.

At the end of this procedure, the matrix $\mathbf{A}_L \triangleq \mathbf{T}(\{U'_{Bx}, x_1, \dots, x_{j_L}\}, \{\mathcal{R}_y^{(j_1)} - y_1 - \dots - y_L, U_{By} - U'_{By}\})$ has rank $|U'_{Bx}| - L$ and the required column set U_{By} . Now to create the matrix $\mathbf{T}(U_{Bx}, U_{By})$ we may need to add to \mathbf{A}_L some additional rows. From Lemma 4 adding these rows cannot increase the rank of the matrix as they belong in the span of $\mathbf{T}(U'_{Bx}, U_{By})$. This completes our proof. ■

Lemma 6: Let j denote the number of already marked inputs when output y gets marked, and let $\mathcal{R}_y^{(j)}$ denote the received set of used outputs from previous iterations at that time. Then there exists a set of rows $S(y)$ in the set $\{\mathcal{R}_x^{(j)}, x_1, \dots, x_j\}$ such that

$$\begin{aligned} \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, U_{By} - U'_{By}\}) &= |S(y)|, \text{ while} \\ \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, U_{By} - U'_{By}\}) &= |S(y)| - 1. \end{aligned} \tag{16}$$

That is, removing the column y drops the rank of the matrix by one, and makes the rows $S(y)$ LD.

Proof: Consider iteration $K + 1$ and layer i . Assume that the node where an output y in U_y belongs gets visited for the first time. This can happen in two ways:

- *Case 1:* The node $A(y)$ gets visited while we perform an Lx -function at layer $i + 1$ (see examples 3 and 4). Note that since we have arrived at layer $i + 1$, we have identified at layer i an edge (x', y') that is LI from the K edges identified from previous iterations.
- *Case 2:* The node $A(y)$ gets visited when we find an edge (x', y') in layer i with $\text{rank}\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)}, y'\}) = K + 1$ (see examples 2 and 3).

The arguments in these two cases are very similar, and we treat them together. In both cases, at layer i , we start with the $(K + 1) \times (K + 1)$ full rank matrix $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)}, y'\})$. When we remove the column y clearly the resulting $(K + 1) \times K$ matrix has some linearly dependent rows. As a result, a subset of the rows becomes linearly dependent. Define $S(y)$ to be the set of inputs in $\{\mathcal{R}_x^{(j)}, x'\}$ corresponding to the *minimally linearly dependent* rows in the matrix $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y'\})$, where by *minimally linear dependent* we mean that the vectors in the set are linear dependent but any proper subset of them is a linearly independent set of vectors. Note that the inputs in $S(y)$ are exactly the inputs that are going to be visited when the algorithm performs the ϕ -function for output y , as, removing any of the rows in $S(y)$ from the matrix $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y'\})$ results in a full rank $K \times K$ submatrix.

Now, since $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)}, y'\})$ is a full rank matrix then there is no set of rows of this matrix which are linearly dependent. In particular, the rows in $S(y)$ are linearly independent. The matrix $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, y', U_{By} - U_{By'}\})$ contains the full rank submatrix $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, y'\})$ and thus has also rank $|S(y)|$. That is

$$\begin{aligned} \text{rank}\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)}, y'\}) &= \text{rank}\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)}, y', U_{By} - U_{By'}\}) = K + 1, \text{ and} \\ \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, y'\}) &= \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, y', U_{By} - U_{By'}\}) = |S(y)|. \end{aligned} \quad (17)$$

Moreover, from construction,

$$\begin{aligned} \text{rank}\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y'\}) &= K, \text{ and} \\ \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y'\}) &= |S(y)| - 1. \end{aligned} \quad (18)$$

We will next argue that

$$\begin{aligned} \text{rank}\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y', U_{By} - U_{By'}\}) &= K, \text{ and} \\ \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y', U_{By} - U_{By'}\}) &= |S(y)| - 1. \end{aligned} \quad (19)$$

that is, adding the columns in $U_{B_y} - U_{B_{y'}}$ does not increase the rank.

Let y_0 be a column in $U_{B_y} - U_{B_{y'}}$, and consider the matrix $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y', y_0\})$. If this square matrix has rank $K + 1$, then the rows $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y', y_0\})$ must be LI. Since the rows $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y'\})$ are LD, there exists row $x_0 \in S(y)$ with a nonzero value in the column y_0 . But when we run the ϕ -function, as we already mentioned, all inputs in $S(y)$ including x_0 are visited and explored. Thus if the $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y', y_0\})$ were full rank, the output y_0 would get marked and not appear in $U_{B_y} - U_{B_{y'}}$. We conclude that for every y_0 in $U_{B_y} - U_{B_{y'}}$, the column $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, y_0)$ belongs in the span of the columns $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y'\})$, and thus, the matrix $\mathbf{T}(\{\mathcal{R}_x^{(j)}, x'\}, \{\mathcal{R}_y^{(j)} - y, y', U_{B_y} - U_{B_{y'}}\})$ has rank K .

Next note that, the rows in matrix $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y'\})$ do not belong in the span of the LI rows $\mathbf{T}(\mathcal{R}_x^{(j)} - S(y), \{\mathcal{R}_y^{(j)} - y, y'\})$. Let y_0 be a column in $U_{B_y} - U_{B_{y'}}$. Clearly, the rows $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y', y_0\})$ do not belong in the span of the LI rows $\mathbf{T}(\mathcal{R}_x^{(j)} - S(y), \{\mathcal{R}_y^{(j)} - y, y', y_0\})$. Thus, if the rows $\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y', y_0\})$ were LI, the matrix $\mathbf{T}(\mathcal{R}_x^{(j)}, \{\mathcal{R}_y^{(j)} - y, y', y_0\})$ would have rank $K + 1$, which is not possible from our previous argument. We conclude that $\text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y', U_{B_y} - U_{B_{y'}}\}) = |S(y)| - 1$. We have thus proved that

$$\begin{aligned} \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, y', U_{B_y} - U_{B_{y'}}\}) &= |S(y)|, \text{ while} \\ \text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, y', U_{B_y} - U_{B_{y'}}\}) &= |S(y)| - 1. \end{aligned} \tag{20}$$

Removing the column y' from the last equation, we also get that

$$\text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, U_{B_y} - U_{B_{y'}}\}) \leq |S(y)| - 1. \tag{21}$$

We now distinguish two cases:

- 1) $S(y) \subseteq \mathcal{R}_x^{(j)}$, i.e., the set of rows $S(y)$ does not contain x' . Then

$$\text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, U_{B_y} - U_{B_{y'}}\}) = |S(y)| - 1, \tag{22}$$

since these rows are LI, and (20)-(22) imply (16).

- 2) $x' \in S'(y)$, we have two subcases:

- a) if $\text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, U_{B_y} - U_{B_{y'}}\}) = |S(y)|$, this together with (21) implies (16).
- b) if $\text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)}, U_{B_y} - U_{B_{y'}}\}) = |S(y)| - 1$, then given (20) the column y' does not belong in the span of the columns $\{\mathcal{R}_y^{(j)}, U_{B_y} - U_{B_{y'}}\}$. Similarly, again from (20), the column y does not belong in the span of the columns $\{\mathcal{R}_y^{(j)} - y, y', U_{B_y} - U_{B_{y'}}\}$. We conclude that $\text{rank}\mathbf{T}(S(y), \{\mathcal{R}_y^{(j)} - y, U_{B_y} - U_{B_{y'}}\}) = |S(y)| - 2$, and thus $\text{rank}\mathbf{T}(S(y) -$

$x', \{\mathcal{R}_y^{(j)} - y, U_{By} - U_{By'}\} \leq |S(y)| - 2$. But $\text{rankT}(S(y) - x', \{\mathcal{R}_y^{(j)}, U_{By} - U_{By'}\}) = |S(y)| - 1$. For the set $S'(y) = S(y) - x'$, the claim in (16) follows. ■

E. Algorithm Complexity

Proposition 4: The complexity of the algorithm in Table I is $O(C^5(O_{total} + \Delta_I I_{total}))$, where C is the capacity of the network, I_{total} equal the total number of inputs and O_{total} is the total number of outputs in the network.

Proof: At iteration K , the complexity of the function “FindL(**T**)” is $O(K^3)$, “Match(**T**)” is $O(K^3)$, to find which inputs to visit with the ϕ -function is $O(K^3)$, and the rank calculations are $O(K^3)$. When we visit each input we will perform at most Δ_I rank calculations, where Δ_I is the maximum outdegree of an input. This results in complexity $O(\Delta_I K^3)$. Moreover, we will perform the “FindL” function at most once for every input. Performing the “FindL” function at the K iteration might result in at most K inputs to be revisited. For each of the revisited inputs, the associated complexity will be $O(\Delta_I K^3)$. Thus, the total complexity when visiting each input is $O(\Delta_I K^4)$. These operations will be repeated at most I_{total} times. An upper bound for I_{total} is $|E|$, where E is the set of all edges in the network, but this bound might be very loose, if the inputs have small outdegree. To conclude, examining the inputs results in complexity of $O(\Delta_I K^4 I_{total})$. When each output is marked, we will perform exactly once the ϕ -function. Thus, this function will be performed at most once for every output (if the output gets marked during the iteration), and contributes complexity $O(K^4 O_{total})$, where (again, a loose upper bound for O_{total} is $|E|$). After C iterations the total complexity is $O(C^5(O_{total} + \Delta_I I_{total}))$. ■

IV. CONCLUSIONS

In this paper we develop a polynomial time algorithm for unicast connections that allow to achieve the min-cut capacity in networks of linear deterministic channels over a finite field \mathbf{F}_q . Such networks have recently found applicability as approximate models for wireless Gaussian networks, by modeling broadcasting and interference through linear operations over a finite field. Our scheme allows to identify the min-cut value in polynomial time, and to achieve this value using very simple one symbol mapping operations at the intermediate network nodes.

REFERENCES

- [1] K. Menger, "Zur allgemeinen Kurventheorie," *Fund. Math.* vol. 10, pp. 95-115, 1927.
- [2] L. R. Ford, Jr. and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399-404, 1956.
- [3] P. Elias, A. Feinstein, and C. E. Shannon, "Note on maximum flow through a network," *IRE Trans. Information Theory*, vol. 2, pp. 117-119, 1956.
- [4] S. Avestimehr, S. N. Diggavi and D. N. C. Tse, "Wireless network information flow", *Proceedings of Allerton Conference on Communication, Control, and Computing*, Illinois, September 2007. See http://licos.epfl.ch/index.php?p=research_projWNC
- [5] S. Avestimehr, S. N. Diggavi and D. N. C. Tse, "A deterministic approach to wireless relay networks", *Proceedings of Allerton Conference on Communication, Control, and Computing*, Illinois, September 2007. See http://licos.epfl.ch/index.php?p=research_projWNC
- [6] S. Avestimehr, S. N. Diggavi and D. N. C. Tse, "Approximate capacity of gaussian relay networks", *IEEE Symposium on Information Theory (ISIT)*, Toronto, July 2008.
- [7] R. Ahlswede, N. Cai, S-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1204-1216, July 2000.
- [8] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egnér, K. Jain and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Trans. Inform. Theory*, vol. 51, no. 6, pp. 1973-1982, 2005.
- [9] N. Harvey, "Deterministic network coding by matrix completion," MS Thesis, MIT 2005.
- [10] T. Cover and J. Thomas, "Elements of Information Theory", Wiley 2006.
- [11] A. Amduruz, C. Fragouli, "Combinatorial algorithms for wireless information flow", SODA 2009.
- [12] M. Goemans, S. Iwata and R. Zenklusen, "An algorithmic framework for wireless information flow", Allerton 2009.

Algorithm III.1: SET OF FUNCTIONS E_A AND $E_x(\cdot)$

```

{(T, F)} =  $E_A(G, \mathcal{P}, \mathcal{M}, A)$ 
if  $\mathcal{M}(A) == T$  return (F)
  else
  {  $\mathcal{M}(A) = T$ 
  {  $U \leftarrow \{\text{used edges in } L(A)\text{-layer cut}\}, U_x \leftarrow \{x_i \in U\}, U_y \leftarrow \{y_j \in U\}, \mathcal{X} \leftarrow \{x_i \in A$ 
  {  $\forall x_i \in \mathcal{X},$  if  $x_i \notin U_x$  and  $\mathcal{M}(x_i) == F$  and  $E_x(G, \mathcal{P}, \mathcal{M}, x_i) == T$  return (T)
  return (F)

{(T, F)} =  $E_x(G, \mathcal{P}, \mathcal{M}, x_i)$ 
if  $\mathcal{M}(x_i) == T$  return (F)
  else
  {  $\mathcal{M}(x_i) = T$ 
  {  $\forall y_j : (x_i, y_j) \in E$ 
  { if  $ML(x_i) == F\%$  (we perform this function only once per input)
  {  $ML(x_i) = T$ 
  {  $L_x = \text{FindL}(\mathbf{T}(\{U_x, x_i\}, U_y))$ 
  {  $\forall x_k \in L_x$ 
  {  $\text{Match}(\mathbf{T}(\{L_x - x_k, x_i\}, L_y))$ 
  {  $\text{Update}(\mathcal{P})$ 
  { if  $\mathcal{M}(A(x_k)) == F$ 
  {  $\forall y_k \in A(x_k)$  perform  $\phi\text{-function}(y_k)$  (see description following)
  { if  $E_A(G, \mathcal{P}, \mathcal{M}, A(x_k)) == T$  return (T)
  { else { if  $\mathcal{M}(x_k) == T$  { Set  $\mathcal{M}(x_k) = F$ 
  { if  $E_x(G, \mathcal{P}, \mathcal{M}, x_k) == T$  return (T)
  { Restore( $\mathcal{P}$ )

  { if  $\text{rank}(\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})) = 1 + \text{rank}(\mathbf{T}(U_x, U_y))$ 
  { if  $A(y_j) == \text{Destination}$  return (T)
  { if  $\mathcal{M}(A(y_j)) == F$ 
  {  $\forall y_k \in U_y$  with  $A(y_k) == A(y_j)$  and  $(x_k, y_k) \in U$ 
  {  $\forall x_k \in U_x$ 
  { if  $\mathbf{T}(\{U_x - x_k, x_i\}, \{U_y - y_k, y_j\})$  is full rank
  {  $U_y = \{U_y - y_k, y_j\}, U_x = \{U_x - x_k, x_i\}$ 
  {  $\text{Update}(\mathcal{P})$ 
  { if  $\mathcal{M}(A(x_k)) == F$ 
  {  $\forall y_\ell \in A(x_k)$  perform  $\phi\text{-function}(y_\ell)$ 
  { if  $E_A(G, \mathcal{P}, \mathcal{M}, A(x_k)) == T$  return (T)
  { else
  { if  $\mathcal{M}(x_k) == T$ 
  { Set  $\mathcal{M}(x_k) = F$ 
  { if  $E_x(G, \mathcal{P}, \mathcal{M}, x_k) == T$  return (T)
  { Restore( $\mathcal{P}$ )
  { if  $E_A(G, \mathcal{P}, \mathcal{M}, A(y_j)) = T$  return (T)
  return (F)

```

\mathbf{F}_q	finite field of operation
S	the source node
D	the destination node
Λ	number of network layers
M	maximum number of nodes per layer
$A(x)$	node where input x belongs
$A(y)$	node where output y belongs
$\mathbf{T}(V, W)$	transformation matrix whose rows are labeled with the elements of V and the columns with the elements of W
$ U $	number of identified LI paths in previous iterations
U	set of used channels between layers i and $i + 1$ (we drop the index i for simplicity)
U_x	set of used inputs at layer i corresponding to the channels in U
U_y	set of used outputs at layer $i + 1$ corresponding to the channels in U
$\mathcal{R}^{(i)}$	the set of edges that input x_i perceives as being used from previous iterations
$\mathcal{R}_x^{(i)}$	set of inputs that input x_i perceives as being used from previous iterations
$\mathcal{R}_y^{(i)}$	set of outputs that input x_i perceives as being used from previous iterations
$L_{x_i}(Z)$	the smallest subset of $\mathcal{R}^{(i)}$ in the matrix $\mathbf{T}(\mathcal{R}^{(i)}, Z)$ that contains x_i in its span
W_i	set of visited nodes in the i layer
W'_i	set of unvisited nodes in the i -th layer
U_{Bx}	set of all the inputs of the nodes in W_x
U'_{Bx}	set of all the visited inputs in the layer i which are used, i.e. $U_{Bx'} = U_{Bx} \cap U_x$.
U_{By}	set of all the outputs in W'_{i+1}
U'_{By}	set of all the outputs of U_{By} which are used, i.e., $U_{By'} = U_{By} \cap U_y$.
I_{total}	total number of inputs in the network.
O_{total}	total number of outputs in the network.

TABLE II
SUMMARY OF NOTATION