

# Combinatorial Algorithms for Wireless Information Flow

Aurore Amaudruz\*  
Nokia Research Center  
Lausanne, Switzerland  
aurore.amaudruz@nokia.com

Christina Fragouli†  
Swiss Federal Institute of Technology (EPFL)  
Lausanne, Switzerland  
christina.fragouli@epfl.ch

## 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 binary 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 binary adjacency matrices describing source-destination cuts. However, since there exists an exponential number of cuts, identifying the capacity through exhaustive search becomes infeasible.

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

## 1 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 bit 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 binary 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 bits 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 bits, where each bit is transmitted at a different power level, and assuming that only bits above a deterministic noise threshold will be successfully received. For example, consider a point-to-point

\*The work by Aurore Amaudruz was done at the Swiss Federal Institute of Technology (EPFL) as part of her Master Thesis.

†This work was supported by Fonds National Suisse (FNS) under award No. 200021-103836/1.

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 ( $\alpha$  represents the channel gain in dB scale  $\alpha \leftrightarrow \lceil \log(\text{SNR}) \rceil$ ). The ADT model is obtained by truncating the received signal and assuming that the  $\alpha$  most significant bits (MSB) of  $x$  are *always* above the deterministic noise threshold and received successfully at node  $B$ . The parameter  $\alpha$  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<sup>nd</sup> MSB from node  $B_2$ .

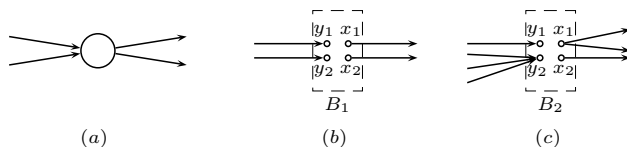


Figure 1: (a) A node in a directed graph, (b) equivalent representation through orthogonal channels, and (c) a node in a network of deterministic channels.

In the ADT model, unlike graphs, channels are no longer orthogonal. Each channel can have multiple inputs and outputs, belonging in different nodes, and the relationship between these inputs and outputs is determined by a set of linear equations. 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 definition 2), 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 lin-

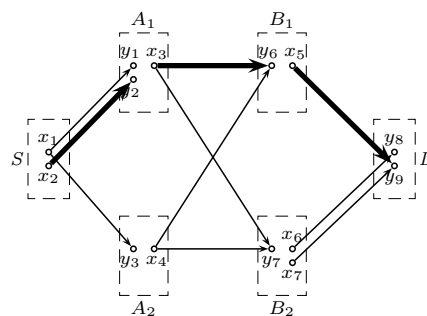


Figure 2: An example of a linear binary deterministic network.

early 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 binary adjacency matrices describing source-destination cuts. However, since there exists an exponential number of cuts, identifying the capacity through exhaustive search becomes infeasible. It would thus clearly be desirable to develop a 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.

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

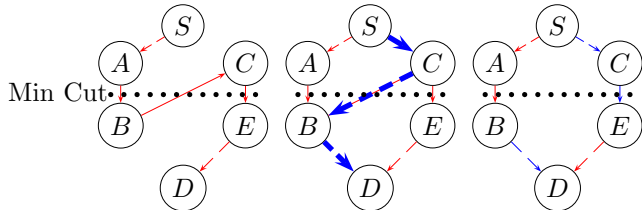


Figure 3: Correcting a “mistake” in the FF algorithm can be thought of as “rewiring paths”.

then complemented by the second path 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” these 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.

The paper is organized as follows. Section 2 introduces our notation and gives an example where constructive use of interference gives significant benefits. Section 3 describes our algorithm and proves that it identifies a minimal value cut. Section 4 concludes the paper.

## 2 Notation and Definitions

We consider a layered network  $G = (V, E)$  with  $|V| = N$  vertices and  $|E|$  edges. The vertices are partitioned into  $\lambda$  layers  $V = V_1 \cup V_2 \cup \dots \cup V_\lambda$ , each layer  $V_i$  having at most  $M$  nodes.  $V_1$  contains only

the source node  $S$ . The channels have inputs  $x_i$  and outputs  $y_i$ , several of which may be contained in each node. The source  $S$  has channel inputs  $\{x_j\}$  connected to the nodes in  $V_2$ .  $V_\lambda$  contains only the destination node  $D$ . The destination has channel outputs  $\{y_j\}$  connected to the nodes in layer  $V_{\lambda-1}$ . Each other node (relay) potentially contains both channel inputs and outputs. If we collect all inputs in the same layer in a row vector  $\mathbf{x}$  and the outputs in the next layer a vector  $\mathbf{y}$  they are related through a linear transformation.

We will denote by  $A(x_i)$  (and  $A(y_i)$ ) the node where a particular input (output) belongs to, and  $\Lambda(A)$  (or  $\Lambda(x_i)$ ,  $\Lambda(y_i)$ ) the layer where a node  $A$  (input of output) belongs to. Edges are of the form  $e = (x_k, y_j)$  with  $\Lambda(y_j) = \Lambda(x_k) + 1$ . We will say that edge  $e$  starts from node  $A(x_k)$  and finishes at node  $A(y_j)$ . We will also say that edge  $e$  contains  $y_j$  and  $x_k$ .

**DEFINITION 1. (Transformation Matrix.)** Given a set of edges  $U = \{(x_i, y_j)\}$ , consider the set of inputs contained in  $U$  described as  $U_x = \{x_i \mid \exists (x_i, y_j) \in U\}$ , and the set of outputs contained in  $U$  described  $U_y = \{y_j \mid \exists (x_i, y_j) \in U\}$ . The transformation matrix  $\mathbf{T}$  associated with the set of edges  $U$  is a binary matrix with rows corresponding to the inputs  $U_x$ , columns corresponding to the outputs  $U_y$ , and value 1 at position  $(i, j)$  if and only if edge  $(x_i, y_j)$  exists.

Note that the transformation matrix is not necessarily square, because for example multiple edges in  $U$  might share the same  $x_i$  or  $y_j$ . Sometimes it will be convenient to associate the transformation matrix not with a set of edges but rather with a set of inputs and outputs. We will then use the notation  $\mathbf{T}(U_x, U_y)$  to specify the row and column set of  $\mathbf{T}$ . We can for example describe the extension of a given transformation matrix  $\mathbf{T}(U_x, U_y)$  by adding a row corresponding to an input  $x_k \notin U_x$  as  $\mathbf{T}(\{U_x, x_k\}, U_y)$  and the extension by adding both an input  $x_k$  and an output  $y_k$  (not already contained in  $U_x$  and  $U_y$ ) as  $\mathbf{T}(\{U_x, x_k\}, \{U_y, y_k\})$ .

**DEFINITION 2. (LI Edges.)** We say that a set of  $K$  edges  $U = \{(x_i, y_j)\}$  are Linearly Independent (LI) if the rank of their transformation matrix  $\mathbf{T}(U_x, U_y)$  equals  $K$ . Otherwise, the set of edges are said to be Linearly Dependent (LD).

A cut is a partition of the nodes into parts  $\mathcal{C}$  and  $\bar{\mathcal{C}}$  with  $S \in \mathcal{C}$  and  $D \in \bar{\mathcal{C}}$ . Let  $U = \{(x, y) : x \in \mathcal{C}, y \in \bar{\mathcal{C}}\}$  be the set of edges that cross the cut (cut edges), the transformation matrix  $\mathbf{T}$  of  $U$  equals the transfer matrix of the cut.

**DEFINITION 3. (Min-Cut value [4, 5].)** The information theoretic value [11] of a cut  $\mathcal{C}$  between

a source-destination pair equals the rank of the transformation matrix  $\mathbf{T}$  associated with the edges crossing the cut. The min-cut value equals the min such value among all possible cuts.

In [4, 5] it is proved that information rate equal to the min-cut value is achievable using information theoretical tools.

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  $\mathcal{C} = V_1 \cup \dots \cup V_j$  and  $\bar{\mathcal{C}} = V_{j+1} \cup \dots \cup V_\lambda$  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.

An S-D path in our setting is a disjoint set of edges  $(e_1, \dots, e_{\lambda-1})$  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.

**DEFINITION 4. (LI Paths.)** We say that  $K$  paths are Linearly Independent (LI) if, for every cut  $\mathcal{C}$ , the rank of the transformation matrix of the used path edges that cross the cut equals  $K$ . That is, if exactly  $K$  edges cross the cut, one from each path, they are LI (according to definition 2).

Note that each  $x_i$  and  $y_j$  can take part in at most one of the  $K$  LI paths; in this case we will say that it is used by that path. We will thus say that a channel input  $x_i$  is *used*, if there exists a path that uses an edge  $(x_i, y_k)$  for any  $y_k$ . Similarly, we will say that a channel output  $y_j$  is *used*, if there exists a path that uses an edge  $(x_k, y_j)$  for any  $x_k$ .

Sometimes we may not be given a set  $U$  of LI edges, but instead, a set of  $U_x$  and  $U_y$  which we are asked to “match” if possible to create LI edges  $(x, y)$  with  $x \in U_x$  and  $y \in U_y$ . We will then use the following known property [9], that we repeat for completeness.

**PROPOSITION 2.1.** 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:* Consider  $\mathbf{T}(U_x, U_y)$  to correspond to the adjacency matrix of a bipartite graph with vertices  $V = (U_x, U_y)$ . Given that  $\mathbf{T}$  is full rank, Hall’s conditions are satisfied on this graph (i.e., each subset  $s$  of rows has at least  $s$  non-zero columns), and thus there exists a perfect matching that gives us the desired edges.  $\square$

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

*Example.* The traditional approach adopted today in wireless networks is that if one or more trans-

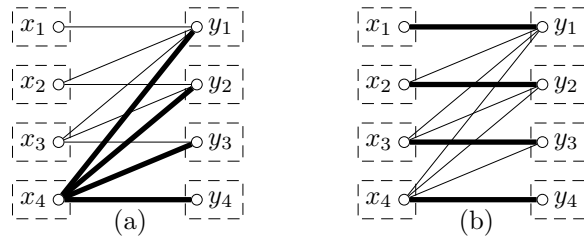


Figure 4: A layer-cut and (a) the traditional approach where interference is treated as noise, (b) the approach where interference is allowed.

mitted 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. 4. Fig. 4(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. 4(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.  $\square$

### 3 The Unicast Algorithm

Our algorithm identifies LI paths. Although we will eventually send one bit through each path, this does not mean that the receiver will receive uncoded information: due to broadcasting and interference, the receiver will still need to solve equations to retrieve the data. What we ensure is that, by the choice of paths, 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.

The algorithm is path augmenting and operates 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. Consider 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 4).

We will use the terminology of “exploring” a

node  $A$  to indicate that we have a path from the source to node  $A$  (LI from the paths in  $\mathcal{P}$ ) and we will now attempt to continue this path from node  $A$  to  $D$  in order to complete  $\mathcal{P}_{K+1}$ . Note that which  $\{y_i\}$  we use to reach a node does not play a role; to explore a node it is sufficient that we arrive at it using any of its  $y_i$ . 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 a function  $\mathcal{M}$  with values  $\{T, F\}$ , 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 our algorithm terminates after a finite number of steps. The algorithm starts by exploring the source node  $S$  and then potentially explores all the nodes and edges in the network.

The main idea is that we are going to allow the algorithm to perform changes inside one layer-cut at a time, that preserve the number of paths going through the nodes. That is, we will allow some type of “rewiring”: we have  $K + 1$  “partial” paths from the source to nodes in layer  $w$ , and  $K$  “partial” paths from nodes from layer  $w + 1$  to the destination. Rewiring refers to that we change the mapping between the starting and finishing paths, while still preserving LI across the  $w$ -layer cut. Let  $U = \{(x_i, y_j)\}$  denote the set of  $K$  used edges in the  $w$ -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  $U$ .

Table 1 summarizes the recursive functions  $E_A$  and  $E_x$  that explore a single node  $A$  and a single input  $x$  respectively, at a layer  $w = \Lambda(A)$ . If we reach the destination we stop, otherwise the functions call themselves to explore the next candidate nodes or inputs. The function  $E_A$  takes as input the layered graph  $G = (V, E)$ , the node  $A \in V$  from where we start (the first time we call the function  $A = S$ ), the set of identified paths in the previous and current iteration (summarized in a structure  $\mathcal{P}$ ), and the set of already marked (visited) nodes and edges (described by the function  $\mathcal{M}$ ). We assume that  $\mathcal{P}$  contains the information for  $U$ ,  $U_x$  and  $U_y$ . The function  $E_A$  returns true if it reaches the destination and false if it fails. Similarly for function  $E_x$ . We next describe the algorithm and illustrate it with the example in Fig. 5.

*Example.* The left plot in Fig. 5 depicts a layer cut. Assume that we are at iteration  $K + 1 = 3$ . During iterations 1 and 2, we have identified the two LI paths that use the bold edges in the figure. Thus,

$$U = \{(x_3, y_1), (x_4, y_3)\}, \quad U_x = \{x_3, x_4\},$$

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

□

To explore a node  $A$ , we sequentially examine all  $x_i \in A$ , and take the following actions.

**1.** If  $x_i$  is already used by a path, i.e.,  $x_i \in U_x$ , do nothing. Note that although node  $A$  will be marked as explored ( $\mathcal{M}(A) = T$ ), this particular  $x_i \in A$  will not be marked ( $\mathcal{M}(x_i) = F$  will remain).

**2.** If  $x_i$  is not used, i.e.,  $x_i \notin U_x$ , then for each  $y_j$ , with  $(x_i, y_j) \in E$ , we distinguish two cases.

**(a)**  $y_j$  is used, i.e.,  $y_j \in U_y$ . 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 3.1 this set can be identified in polynomial time. Let  $L_y = \{y_j \mid (x_i, y_j) \in U, x_i \in L_{x_i}\}$  be the set of outputs of the edges in  $U$  with inputs in  $L_{x_i}$ . Since these edges are LI, the square matrix  $\mathbf{T}(L_{x_i}, L_y)$  has rank  $s$ . Then Clearly, the matrix  $\mathbf{C} \triangleq \mathbf{T}(\{L_{x_i}, x_i\}, L_y)$  has also rank  $s$ . Proposition 3.2 proves that removing any one of the rows of  $\mathbf{C}$  (corresponding to an employed  $x_k \in L_{x_i}$ ) still results in a full rank matrix. This implies that, using Proposition 2.1, we can use  $x_i$  to substitute any of the already employed  $x_k \in L_{x_i}$  that are LD with  $x_i$ , 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. Note that if  $A(x_k)$  is already marked as explored, we do not need to visit it again; however, if the particular input  $x_k$  is not marked, we will still attempt to find paths starting from  $x_k$ .

In this step we encountered some LD relationships. For each input  $x \in U$  we will keep a set of associated inputs  $R_x$ , whose purpose we will discuss later; in this set we will add all inputs that occurred in the same set  $L_{x_i}$  as  $x$  for any  $x_i$ .

*Example.* In Fig. 5 assume that we have reached node  $A_4$  and we examine the edge  $(x_5, y_1)$ :

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

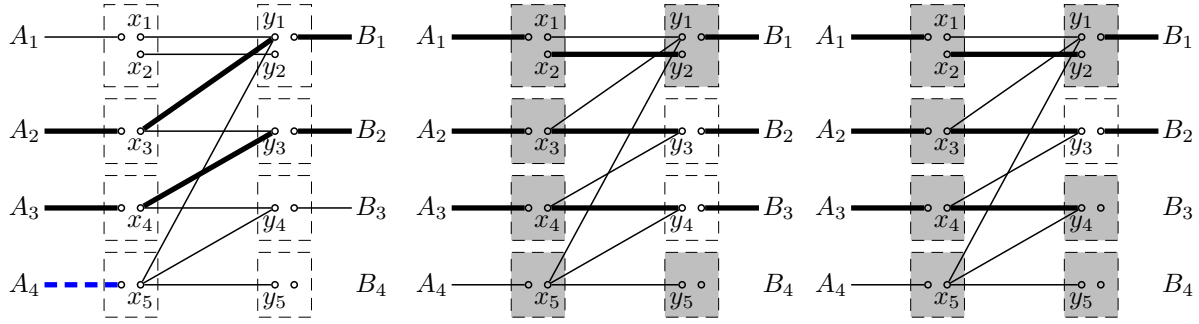


Figure 5: Left: paths before rewiring, Middle: paths after the algorithm runs, Right: identified cut if nodes  $B_3$  and  $B_4$  had no outgoing edges.

Then  $\text{rank}(\mathbf{T}(\{U_x, x_5\}, U_y)) = 2$  and  $L_{x_5} = \{x_3, x_4\}$ . We can attempt to substitute each of the  $x \in L_{x_5}$  with  $x_5$ .

- If we first substitute  $x_3$ , we mark  $A_2$  and find a matching:  $\{(x_5, y_1), (x_4, y_3)\}$ . We call  $E_A(G, \mathcal{P}, \mathcal{M}, A_2)$  and proceed. Assume that this function returns F, i.e., we failed to connect  $A_2$  to the destination.
- We then substitute  $x_4$ , mark  $A_3$  and find another matching:  $\{(x_5, y_1), (x_3, y_3)\}$ . We call  $E_A(G, \mathcal{P}, \mathcal{M}, A_3)$ . Again assume we fail to find a path to the destination.  $\square$

(b)  $y_j$  is not used. Consider the  $(K + 1) \times (K + 1)$  binary matrix  $\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})$  associated with the used edges and the new edge  $(x_i, y_j)$ .

- (i) If this matrix is not full rank, do nothing. In this case there exists at least one edge  $(x_i, y_k)$  with  $y_k \in U_y$ . Indeed, otherwise if the last row of  $\mathbf{T}$  had zeroes everywhere appart the last element, we would have that  $\det(\mathbf{T}(\{U_x, x_i\}, \{U_y, y_j\})) = \det(\mathbf{T}(U_x, U_y)) \neq 0$ . Since such an edge exists, case (2a) will at some point be executed, and as we will see there are no additional actions needed.

*Example.* In Fig. 5, examine the edge  $(x_5, y_4)$ :

$$\mathbf{T}(\{U_x, x_5\}, \{U_y, y_4\}) = \begin{matrix} & y_1 & y_3 & y_4 \\ x_3 & 1 & 1 & 0 \\ x_4 & 0 & 1 & 1 \\ x_5 & 1 & 0 & 1 \end{matrix}$$

Because  $\text{rank}(\mathbf{T}(\{U_x, x_5\}, \{U_y, y_4\})) = \text{rank}(\mathbf{T}(U_x, U_y)) = 2$  we do not proceed further.  $\square$

- (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  $A \triangleq A(y_k) = A(y_j)$ , perform what we call the  $\phi$ -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  $A$ . This path uses an edge  $(x_k, y_k) \in U$  to reach node  $A = A(y_k)$ . We can then use our newly identified partial path that uses the edge  $(x_i, y_j)$  to reach from the source the node  $A = A(y_j)$ , and “connect” this new partial path with the existing partial path from  $A$  to destination. Thus, we no longer need to use the edge  $(x_k, y_k)$ , and we have an available  $S - A(x_k)$  partial path to attempt to complete. Then the  $\phi$ -function does the following:

Remove  $(x_k, y_k)$  from the set of used edges. If  $A(x_k)$  is not marked as visited, explore  $A(x_k)$ . If  $A(x_k)$  is marked as visited, then:

-If input  $x_k$  is not marked, we examine it. Note that since it is not marked, it has not blocked the use of any edge.

-If the input  $x_k$  is marked, this implies that, since the edge  $(x_k, y_k)$  was used, there exists an input  $x_\ell \notin U_x$  and an associated edge that could not be used due to LD, and  $L_{x_\ell}$  contained  $x_k$ . Since the edge  $(x_k, y_k)$  will now be removed from the set of used edges, it may be possible to use now edges that we could not before. To achieve that, we need to be able to examine again all inputs that belong in  $\{L_{x_\ell}\}$ , although they have been examined before. This is why, for every input  $x_\ell$  we have kept track of the set of related inputs  $R_{x_\ell}$ . When the edge  $(x_k, y_k)$  is removed from  $U$ , the set  $R_{x_k}$  will contain all  $x$  that potentially need to be examined again. We can thus unmark all these inputs, and attempt to find a path starting from  $x_k$ . Note that the  $\phi$ -function will be executed at most  $M$  times, where  $M$  is the maximum number of vertices in a layer.

Intuitively, removing the edge  $(x_k, y_k)$  from  $U$  alters the subspace that the rows in  $U_x$  span. If the edge  $(x_k, y_k)$  has not prohibited the use of any other edge, its removal from  $U$  will not affect LI conditions. If the edge  $(x_k, y_k)$  had prohibited, due to LD, the use of another edge, the set  $R_k$  keeps track of the set of inputs that executing the  $\phi$ -function causes to re-examine.

**Algorithm 3.1:** 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
  {  $\mathcal{M}(A) = T$ 
     $U \leftarrow \{\text{used edges in } L(A)\text{-layer cut}\}, U_x \leftarrow \{x_i \in U\}, U_y \leftarrow \{y_j \in U\}, \mathcal{X} \leftarrow \{x_i \in A\}$ 
     $\forall x_i \in \mathcal{X}, \text{ if } x_i \notin U_x \text{ and } M(x_i) = F \text{ and } E_x(G, \mathcal{P}, \mathcal{M}, x_i) = T$  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  $y_j \in U_y$ 
      {  $(L_x, L_y) = \text{FindL}(\mathbf{T}(\{U_x, x_i\}, U_y))$ 
         $\forall x_k \in \{L_x\}, R_{x_k} \leftarrow \{R_{x_k} \cup L_x\}$ 
         $\forall x_k \in L_x$ 
        {  $\text{Match}(\mathbf{T}(\{L_x - x_k, x_i\}, L_y))$ 
          Update( $\mathcal{P}$ )
          if  $\mathcal{M}(A(x_k)) = T, \text{ if } \mathcal{M}(x_k) = F \text{ and } E_x(G, \mathcal{P}, \mathcal{M}, x_k) = T$  return (T)
          else if  $E_A(G, \mathcal{P}, \mathcal{M}, A(x_k)) = T$  return (T)
          Restore( $\mathcal{P}$ )
        }
      }
    if  $y_j \notin U_y$ 
      { 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  $\mathcal{M}(A(y_j)) = F$ 
          { if  $E_A(G, \mathcal{P}, \mathcal{M}, A(y_j)) = T$  return (T)
             $\forall y_k \in U_y \text{ with } A(y_k) = A(y_j) \text{ and } (x_k, y_k) \in U$ 
            {  $U_y \leftarrow U_y - y_k, U_x \leftarrow U_x - x_k, U \leftarrow U - (x_k, y_k)$ 
              Update( $\mathcal{P}$ )
              if  $\mathcal{M}(A(x_k)) = F \text{ if } E_A(G, \mathcal{P}, \mathcal{M}, A(x_k)) = T$  return (T)
              else {  $\forall x \in R_{x_k}, \mathcal{M}(x) = F$ 
                if  $E_x(G, \mathcal{P}, \mathcal{M}, x_k) = T$  return (T)
              }
            }
          }
        }
      }
    }
  }
return (F)

```

Table 1: The functions  $E_A(\cdot)$  and  $E_x(\cdot)$  are executed by the algorithm at each explored node or edge. The function “Match( $\mathbf{T}$ )” finds a perfect matching in the bipartite graph defined by matrix  $\mathbf{T}$  as described in proposition 2.1. The function “FindL( $\mathbf{T}$ )” finds the smallest set of rows that are LD with the last row of  $\mathbf{T}$  as described in proposition 3.1. The function “Update( $\mathcal{P}$ )” keeps track of the current wiring of identified paths (which may change either by the execution of the match function, or by the  $\phi$ -function), while “Restore( $\mathcal{P}$ )” restores  $\mathcal{P}$  to the value before the last update. The set  $R_{x_k}$ , initialized to the empty set at the beginning of each iteration, keeps for every input  $x_k$  the current set of identified inputs that are in a LD relationship with  $x_k$ . The labeling function  $M(\cdot)$  equals T if a node (or edge) is already explored, in which case it is not explored again, and F otherwise.

*Example.* In Fig. 5 examine the edge  $(x_5, y_5)$ :

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

Since  $\text{rank}(\mathbf{T}(\{U_x, x_5\}, \{U_y, y_5\})) =$

$\text{rank}(\mathbf{T}(U_x, U_y)) + 1 = 3$ , we mark  $B_4$  and update  $\mathcal{P}$ . In  $B_4$  there is no need to perform the  $\phi$ -function. We call  $E(G, \mathcal{P}, \mathcal{M}, B_4)$ . Assume it returns F, fails to find a path to the destination.

We have thus failed to find a path when exploring  $A_4$ . Now suppose that, through some differ-

ent path not depicted in Fig. 5, we reach node  $A_1$  and explore  $x_1$  and  $x_2$ . Assume we start by edge  $(x_2, y_2)$ . We can use this edge to reach  $B_1$ . We call  $E_A(G, \mathcal{P}, \mathcal{M}, B_1)$  and assume we fail. But we now can execute the  $\phi$ -function. We remove the edge  $(x_3, y_1)$  from the set of used edges  $U$ , and substitute it with the edge  $(x_2, y_2)$ . We unmark the inputs in the set  $R_{x_3}$  (but not the nodes  $A_2$  and  $A_3$ ) and attempt to proceed from input  $x_3$ . Attempting to use the edge  $(x_3, y_3)$  will eventually result in the set of paths depicted in the middle Fig. 5.

For the sake of illustration, assume now that we are in the case where we can't find a path to the destination from  $B_3$  as described in the right Fig. 5. When arriving at node  $A_4$ , the algorithm marks  $A_4, B_4, A_2$  and  $A_3$ . It is not able to proceed any further. When the algorithm arrives in  $A_1$ , it marks  $B_1$  when exploring edge  $(x_2, y_2)$ , and when trying to rewire the edges, it is able to reach  $B_3$  and marks it as well. The set of marked nodes defines a minimum cut (of value two in our case) as represented in the right Fig. 5.  $\square$

**PROPOSITION 3.1.** *Let  $\mathbf{T}(U_x, U_y)$  be a full rank  $K \times K$  matrix and  $\mathbf{v} = \mathbf{T}(x_i, U_y)$  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  $O(K^4)$  operations.*

*Proof:* For each  $x_j \in U_x$ , if  $\text{rank}(\mathbf{T}(\{U_x - x_j, x_i\}, U_y)) = K$  then  $x_j \in L_{x_i}$ . The idea is the vectors in  $\{U_x - x_j, x_i\}$  will have rank less than  $K$  only if they include all the vectors in  $L_{x_i}$  and  $x_i$ . This algorithm takes  $O(K^4)$  operations.  $\square$

**PROPOSITION 3.2.** *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  $L_{x_i}$ ,  $\mathbf{x}_i = \sum_{j \in L_{x_i}} \mathbf{x}_j$ , otherwise, we could have found a smaller set to replace  $L_{x_i}$ . Thus, for any  $x_k \in L_{x_i}$ ,

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

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.  $\square$

Our main result is the following theorem.

**THEOREM 3.1.** *The unicast algorithm identifies  $C$  LI paths, where  $C$  is the min-cut value between the source-destination pair in a linear deterministic network.*

*Proof.* To prove that our algorithm works in arbitrary networks, we need three statements:

1. An arbitrary network through time-expansion can be converted into a layered network for the purpose of determining the intermediate node operations.
2. The paths that the algorithm identifies are LI.
3. When the algorithm stops, the number of paths equals the identified cut value.

The first part is proved in [4, 5].

For the second part, note that by construction, all used edges inside a layer cut are LI. Since edges can only be LD with other edges in the same layer cut, we are done.

For the third part, assume that we have found  $K$  paths, and at the  $K + 1$  iteration the algorithm stops. We will argue that then  $C = K$ . Consider the set of nodes that are marked as reached. Since these do not include the destination, they define a cut between  $S$  and  $D$ . It is sufficient to show that this cut value equals  $K$ .

Consider the transfer matrix of this cut. Each path needs to cross the cut at least once. We will prove that, in the identified cut, each path crosses the cut exactly once. Thus the matrix will include  $K$  rows corresponding to the inputs used by the  $K$  identified paths. As discussed, we have two cases of cuts, layer cuts and cross cuts. Because layer cuts can be thought of as special cases of cross cuts, we will focus our attention on the latter.

The transfer matrix of a cross cut is block diagonal. Each block  $\mathbf{B}$  has inputs  $\{x_i\}$  from a layer  $w$ , outputs  $\{y_j\}$  from a layer  $w + 1$ , and value 1 for each edge  $(x_i, y_j)$  crossing the cut. More formally, if  $W_x$  is the set of  $x$  on all marked nodes on layer  $w$  and  $W_y$  the set of all  $y$  in not marked nodes in layer  $w + 1$ , the cut transfer matrix will be  $\mathbf{T}(W_x, W_y)$ . Note that, for a broadcasting node, not all outgoing edges need to cross the cut; it is possible that  $x_i \in W_x$ , but not all  $y_k$  with  $(x_i, y_k) \in E$  are in  $W_y$ . Similarly, for a  $y_j$  where interference happens, not all incoming edges need to cross the cut; that is, for a particular  $y_j \in W_y$ , not all existing edges  $(x_k, y_j)$  have  $x_k \in W_x$ . Only the edges connecting the set of nodes that are marked on layer  $w$  and the set of nodes that are not marked on layer  $w + 1$  will cross the cut.

Let  $U_x$  and  $U_y$  denote the set of used inputs and outputs respectively in the layer we are examining. Let  $U_B \subseteq U$  denote the set of used edges appearing in block  $\mathbf{B}$ , i.e.,  $U_{Bx} = U_x \cap W_x$ ,  $U_{By} = U_y \cap W_y$ . If  $U_{Bx}$  has size  $K_i$  (with  $\sum K_i = K$  over all blocks), we will prove that block  $B$  has rank  $K_i$ , and thus conclude that the transfer matrix of the

overall cut has rank  $K$ . That is, we will show that  $\text{rank}(\mathbf{T}(W_x, W_y)) = \text{rank}(\mathbf{T}(U_{Bx}, U_{By}))$ , i.e., the  $x \in W_x - U_{Bx}$ , and  $y \in W_y - U_{By}$  do not increase the rank of  $\mathbf{B}$ .

Our argument proceeds as follows. Assume that  $x_i \in W_x$ . We will distinguish between the cases where  $x_i$  is used, and the case where it is not.

**I)**  $x_i \in U_x$ . Assume an  $x_i$  is used. Assume  $x_i$  is a broadcasting node with  $m > 1$  outgoing edges, say  $(x_i, y_1), \dots, (x_i, y_m)$ , and let  $(x_i, y_1)$  be the used edge. Assume that the remaining edges  $(x_i, y_k)$  are independent from all other used edges in layer  $w$  and thus the use of  $(x_i, y_1)$  is the only reason that prevents these edges from being used. We need to ensure that, it is not possible for an edge  $(x_i, y_k)$  to cross the cut unless the used  $(x_i, y_1)$  crosses it as well.

When  $x_i$  is examined,  $A(x_i)$  will be marked while  $A(y_1), \dots, A(y_m)$  will not. Assume there exists an alternative potential path that arrives to node  $A(y_1)$  and marks this node. Then, the  $\phi$ -function will be applied to node  $A(y_1)$ , and all nodes  $A(y_2), \dots, A(y_m)$  will also be visited. Thus these edges will not be included in the cut.

Assume now that some of the edges  $(x_i, y_k)$ ,  $k = 2 \dots m$ , are not independent to other used edges in this layer. Then, although the input  $(x_i, y_1)$  may no longer be used (because for example we arrived at node  $A(y_1)$  and applied the  $\phi$ -function) we may still not be able to visit  $A(y_k)$  because of LD - but since now the LD constraints have become active, we go to another case of our algorithm.

**II)**  $x_i \notin U_x$ . We are interested in the  $x_i \in W_x$ . Clearly, since  $x_i \notin U_x$ , we have that  $x_i \in W_x - U_{Bx}$ .

We will use the following notation. Assume that  $\text{rank}(\mathbf{T}(\{U_x, x_i\}, Z)) = \text{rank}(\mathbf{T}(U_x, Z))$  for some set of columns  $Z$ . Define  $L_{x_i}(Z) \subseteq U_x$  to be the smallest subset of  $U_x$  that contains  $x_i$  in its span, i.e.,  $\text{rank}(\mathbf{T}(\{L_{x_i}(Z), x_i\}, M)) = \text{rank}(\mathbf{T}(L_{x_i}(Z), Z))$ . We have already encountered the set  $L_{x_i} = L_{x_i}(U_y)$ .

For our specific  $x_i$ , decompose the indices  $W_y$  in the following 4 nonoverlapping parts:  $W_y = [U_{By}, W_i, W_L, W_0]$ . Here

- $U_{By}$  contains all the used  $y$ 's,
- $W_i$  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  $y_k \in W_y$  that have at least one 1 in each column (i.e., the set of all  $y$  columns where at least one edge  $(x_k, y_k)$  with  $x_k \in L_{x_i}(W_y)$  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  $\{L_{x_i}(W_y), x_i\}$ ).

We will first prove the following property:

**Property 1:** For all  $x_i \in W_x - U_{Bx}$  it holds that

$$(3.1) \quad \begin{aligned} L_{x_i} &= L_{x_i}(U_y) \stackrel{(a)}{=} L_{x_i}(\{U_y, W_i\}) \\ &\stackrel{(b)}{=} L_{x_i}(\{U_y, W_L\}) \stackrel{(c)}{=} L_{x_i}(\{U_y, W_0\}). \end{aligned}$$

*Proof.*

(a) We will first show that  $L_{x_i} = L_{x_i}(U_y) = L_{x_i}(U_y, W_i)$ , i.e.,

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

From definition,

$$(3.2) \quad \mathbf{T}(x, U_y) = \sum_{x_i \in L_{x_i}(U_y)} \mathbf{T}(x_i, U_y)$$

and

$$(3.3) \quad \mathbf{T}(x, \{U_y, W_i\}) = \sum_{x_i \in L_{x_i}(U_y, W_i)} \mathbf{T}(x_i, \{U_y, W_i\})$$

Expurgating from both sides of (3.3) the columns of  $W_i$  results in an equation that still holds for the expurgated vectors and has only columns corresponding to  $U_y$ . From LI of all vectors  $\mathbf{T}(x, U_y)$ ,  $x \in U_x$ , none of these expurgated vectors is identically zero. Moreover, from minimality of  $L_{x_i}(U_y)$  the expansion (3.2) is unique. We thus conclude that  $L_{x_i} = L_{x_i}(U_y) = L_{x_i}(U_y, W_i)$ .

(b) We will now argue that  $L_{x_i} = L_{x_i}(U_y) = L_{x_i}(\{U_y, W_L\})$ . Consider a specific  $y_k \in W_L$  that has one in 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  $x_i$ . That is, the set of used  $x$ 's will now be  $U'_x \triangleq \{U_x, x_i\} - x_k$ . Because  $L_{x_i}(U_y) = L_{x_k}(U_y)$ , applying the same argument as in (a) we get the desired result.

(c) Clearly it also holds that

$$\text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, U_y)) = \text{rank}(\mathbf{T}(L_{x_i}, \{U_y, W_0\})).$$

We thus conclude that

$$(3.4) \quad \begin{aligned} \text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, \{U_y, W_i, W_L, W_0\})) &= \\ &= \text{rank}(\mathbf{T}(L_{x_i}, \{U_y\})). \end{aligned}$$

□

Assume now we expurgate some of the columns in  $U_y$ , to create  $U_{By} \subseteq U_y$ .

From minimality of  $L_{x_i}(U_y)$ , expurgation of columns does not alter the above relationship - it may simply reduce some rows to all zero rows. What is important for us, however, is that, no matter which columns are removed, including the row corresponding to  $x_i$  does not increase the rank. We have thus proved that:

$$\begin{aligned} \text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, W_y)) &= \\ &= \text{rank}(\mathbf{T}(\{L_{x_i}, x_i\}, \{U_{By}, W_i, W_L, W_0\})) = \\ &= \text{rank}(\mathbf{T}(L_{x_i}, W_y)). \end{aligned}$$

We next need to prove that  $L_{x_i} \subseteq U_{Bx}$  (to be exact, that the nonzero rows in  $L_{x_i}$  appearing in (3.5) belong in  $U_{Bx}$ ). To do that, we will go through the algorithm steps for the cases where  $x_i \notin U_x$ .

- Assume  $x_i$  is not used,  $y_j$  is not used, and LI holds. Then the edge  $(x_i, y_j)$  will not appear in the cut, since  $A(y_j)$  will be visited and marked.

- Assume  $x_i$  is not used,  $y_j$  is not used, but LD holds. Then the vector  $\mathbf{T}(x_i, U_y)$  has at least one nonzero value, and the algorithm will visit all  $A(x)$  with  $x \in L_{x_i}(U_y)$ . Consider all the edges  $(x, y)$  with  $x \in L_{x_i}(U_y)$ , and  $A(x)$  marked by the algorithm. If the node  $A(y)$  are not marked, the edges  $(x, y)$  will be included in the cut. Assume now that for one of these edges, say  $(x_k, y_k)$ ,  $A(y_k)$  gets marked, and thus the edge  $(x_k, y_k)$  does not get included in the cut. But then, the  $\phi$ -function will be performed at node  $A(y_k)$ . From definition of the set  $L_{x_i}$ ,

$$\text{rank}(\mathbf{T}(L_{x_i}, \{L_y - y_k, y_j\})) = \text{rank}(\mathbf{T}(L_{x_i}, L_y))$$

and thus the node  $A(y_j)$  will be visited by the algorithm and the edge  $(x_i, y_j)$  will also not be included in the cut.

- Assume an  $x_i$  is not used but  $y_j$  is. Similar arguments as in the previous case apply.

Intuitively speaking, if an edge  $(x_i, y_j)$  cannot be used due to LD with a set of edges  $\{(x_k, y_k)\}$ , with  $x_k \in L_{x_i}$ , substituting  $x_i$  for  $x_k$  ensures that all  $A(x_k)$  are marked as visited. The  $\phi$ -function on the other hand ensures that, if among the set of edges  $\{(x_k, y_k)\}$ , with  $x_k \in L_{x_i}$ , any node  $A(y_k)$  containing an edge output is marked, so do all other output nodes including  $A(y_j)$ , and thus none of the edges  $\{(x_i, y_j), (x_k, y_k)\}$  appear in the cut.

Finally note that, due to the use of the  $\phi$ -function, when the algorithm stops, it not possible to have an edge  $(x_k, y_k) \in U$  where  $A(y_k)$  is marked while  $A(x_k)$  is not. This shows that each path can only cross the identified cut once.  $\square$

**PROPOSITION 3.3.** *The complexity of the algorithm is  $O(M \cdot |E| \cdot C^5)$ .*

*Proof:* Let  $C$  be the capacity of the network. At iteration  $K$ , the complexity of the function “FindL( $\mathbf{T}$ )” is  $O(K^4)$ , “Match( $\mathbf{T}$ )” is  $O(K^3)$ , and the rank calculations are  $O(K^3)$ . For each edge, we will in the worst case perform all these operations, at most  $M$  times, where  $M$  is the maximum number of nodes per layer, if the  $\phi$ -function is called and unmarks the edge. Note that the  $\phi$ -function cannot be called more than  $M$  times.  $\square$

## 4 Conclusions

In this paper we develop polynomial time algorithms for unicast connections that allow to achieve the min-cut capacity in networks of deterministic channels. Such networks have recently found applicability as approximate models of wireless Gaussian networks. Our scheme allows to identify the min-cut value in polynomial time, and to achieve this value using very simple one bit operations at the intermediate network nodes.

## Acknowledgments

We would like to thank Suhas Diggavi for useful discussions. We would also like to thank Soheil Mohajer and the anonymous reviewers for their comments on the paper.

## 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](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](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. Egner, 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] A. Amaxudruz, C. Fragouli, “Multicasting over linear deterministic networks”, *Allerton Conference on Communication, Control, and Computing*, September 2008.
- [11] T. Cover and J. Thomas, “Elements of Information Theory”, Wiley 2006.