Polynomial-Time Construction of Linear Network Coding

Kazuo Iwama[1,⋆], Harumichi Nishimura[2,⋆⋆], Mike Paterson[3,⋆⋆⋆], Rudy Raymond[4], and Shigeru Yamashita[5,†]

[1] School of Informatics, Kyoto University, Japan
iwama@kuis.kyoto-u.ac.jp
[2] School of Science, Osaka Prefecture University, Japan
hnishimura@mi.s.osakafu-u.ac.jp
[3] Department of Computer Science and DIMAP, University of Warwick, UK
msp@dcs.warwick.ac.uk
[4] Tokyo Research Laboratory, IBM Japan, Japan
raymond@jp.ibm.com
[5] Graduate School of Information Science, Nara Inst. of Science & Technology, Japan
ger@is.naist.jp

Abstract. Constructing k independent sessions between k source-sink pairs with the help of a linear operation at each vertex is one of the most standard problems in network coding. For an unbounded k, this is known to be NP-hard. Very recently, a polynomial-time algorithm was given for $k = 2$ [Wang and Shroff, ISIT 2007], but was open for a general (constant) k. This paper gives a polynomial-time algorithm for this problem under the assumption that the size of the finite field for the linear operations is bounded by a fixed constant.

1 Introduction

The max-flow min-cut theorem is a fundamental law for communication networks. So, it was remarkable when Ahlswede, Cai, Li and Yeung showed that this law can be bypassed by *network coding* [2]. They gave a small and nice example called the Butterfly network. As shown in Figure 1, the Butterfly network has two source-sink pairs (s_1, t_1) and (s_2, t_2), and it is easily seen that if we remove a single link, i.e., the one from s_0 to t_0, then there is no path from s_1 to t_1 or from s_2 to t_2 any longer. Therefore, we cannot achieve two disjoint paths for the two source-sink pairs in order to send two bits, x from s_1 to t_1 and y

⋆ Supported in part by Scientific Research Grant, Ministry of Japan, 16092101 and 19200001.
⋆⋆ Supported in part by Scientific Research Grant, Ministry of Japan, 19700011.
⋆⋆⋆ Supported in part by DIMAP (Centre for Discrete Mathematics and its Applications), EPSRC, UK, and by Scientific Research Grant, Ministry of Japan, 16092101.
† Supported in part by Scientific Research Grant, Ministry of Japan, 16092218 and 19700010.

L. Aceto et al. (Eds.): ICALP 2008, Part I, LNCS 5125, pp. 271–282, 2008.

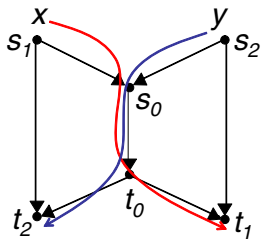

Fig. 1. Butterfly network

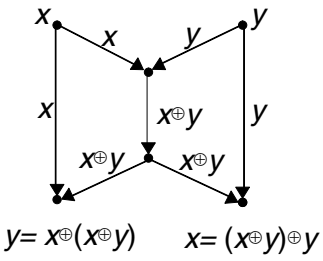

Fig. 2. Coding scheme

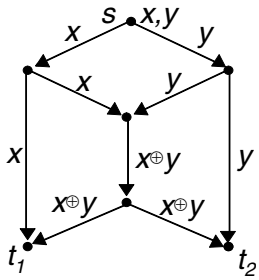

Fig. 3. Multicast example

from s_2 to t_2, simultaneously. However, if we allow "coding" at each vertex, then we can consider, for instance, the protocol in Figure 2 and two bits x and y can now be transmitted simultaneously! After [2], network coding became instantly popular and the large literature has investigated the possibility and applications of network coding (see the network coding home page [11]).

Network coding has several different models, but the two major ones are (i) the *k-source-sink pair model* (also called the multiple-source unicast model) and (ii) the *multicast model*. A typical example of (i) is the butterfly network, where $k = 2$. The latter model has only one source (and many sinks) but usually the source has multiple inputs. For example, the model in Figure 3 has one source s with two inputs x, y, and two sinks t_1 and t_2 which both require x and y in a single round. (An obvious necessary condition is that there must be a sufficient number of links from the source vertex, two in the case of Figure 3, and similarly for each sink.) There is a large literature for this model: for example, (A) Li, Yeung and Cai [18] showed that a network coding exists if and only if a *linear* network coding exists (i.e., the operation at every vertex is a linear combination of inputs on a finite field alphabet), (B) the maximum size of an alphabet in the linear coding can be bounded by a relatively small number [9,10], and (C) such a network coding (if any) can be found in polynomial time [10,7,13].

The k-source-sink pair model, requiring independent sessions between sources and sinks as in the Butterfly, is much more difficult. In fact, this model is essentially the same as the most general model where (not necessarily pairwise) sources and sinks may have multiple inputs and multiple outputs, respectively [4]. Compared to the multicast model, (A′) linear coding is not sufficient, i.e., there exist graphs that do not admit linear network coding but admit vector-linear network coding (vector-linear operations instead of linear operations are allowed at each vertex) [16,19], (B′) the maximum alphabet size is not known even if we only consider linear coding, and (C′) if k is not bounded, then whether a given graph has a linear network coding is NP-complete for any fixed alphabet size [16].

Although (A′) shows a limit of linear coding, linear network coding is still popular because of its simplicity and in fact admits several nice features (e.g., exponentially larger bandwidth) compared to conventional routing (e.g., [1,8]). Therefore, considering (C′) also, our natural question for the k-pair model is

whether there is a polynomial-time algorithm for deciding the possibility of linear network coding for a small constant k, which has been one of the most important open questions in the field.

Very recently there was some important progress toward this goal; Wang and Shroff [22,23] showed that the problem is solvable in polynomial time if $k = 2$. The proof exploits nontrivial graph theoretic properties, which furthermore implies that (i) linear coding is enough and (ii) two is enough for the size of its alphabet. Thus the question was answered almost completely for $k = 2$, but at the same time, the above two facts suggest that the case of $k = 2$ is somewhat special and it is hard to extend their approach to a general (constant) k.

1.1 Our Contribution

In this paper we present an algorithm that decides, given a directed acyclic graph and k source-sink pairs, whether or not a linear network coding exists. It runs in time $O(n^{g(a,k)})$ where n is the number of vertices in the graph and $g(a, k)$ is a function depending only on k and a, the maximum size of the field $\mathbb{F}$ we can use for linear coding. Thus, if both k and a are constants, then this is a polynomial-time algorithm. Unfortunately it is not known if such a (i.e., depending only on k and not on n) is enough for a linear network coding.

Here is the basic idea: we first show, using a similar idea to that in [14], that the given graph can be transformed, with a sacrifice of a quadratic increase in its size, into another graph whose maximum indegree is two. Then the number of different linear operations at each vertex is at most $|\mathbb{F}|^2$, which is constant since we assumed that $|\mathbb{F}|$ is bounded by a constant. Thus far, we have a trivial exponential-time algorithm by considering $(|\mathbb{F}|^2)^{n^2}$ combinations of linear operations for all vertices. To decrease this complexity, we first prove the key lemma stating that the number of "real" coding vertices can be bounded by a constant (only dependent on k). "Real" means that the output state of the gate (vertex) depends on both input states. In other words, in all the other gates the output state depends on only one of the two input states.

Roughly speaking, what remains to be done in those gates is to select one of the two inputs and connect it to the output. It turns out that this is equivalent to finding vertex-disjoint paths between h (again bounded by a constant) pairs of vertices, which is known as the Disjoint Paths Problem. For directed acyclic graphs, the general problem for unrestricted h is still NP-hard, but fortunately polynomial-time algorithms for constant h were found by Fortune et al. [5].

1.2 Related Work

As mentioned in [15], there are many variables to consider in network coding. Here we list some of them which are closely related to this paper.

Alphabet size. The complexity of the problem with respect to the size of the alphabet (the size of the finite field in the case of linear network coding)

is subtle, since if the size is larger we can use more powerful operations but at the same time we need to transmit more information in each round. However, from a practical viewpoint, it is obviously better to use smaller alphabets. In the multicast model, an upper bound of the alphabet size for linear network coding, $O(h)$ where h is the number of sinks, was shown by Ho et al. [9] based on the algebraic framework in [12]. This upper bound (and the algebraic argument in [12,9]) was used in a polynomial-time algorithm by Harvey et al. [7]. Another (in fact, the first) polynomial-time algorithm, also with alphabet size bounded by $O(h)$, was given by Jaggi et al. [10]. On the other hand, the lower bound on the alphabet size of $\Omega(\sqrt{h})$ is shown in [16,21], and deciding the smallest alphabet size is NP-hard [16].

Much less is known for the k-source-sink pair model. There is a network coding that has a doubly-exponential lower bound (in the graph size n) on alphabet size [17]. However, this lower bound example does not admit linear network coding (but admits a vector-linear coding), and its k increases with n. So, the maximum alphabet size, especially whether it depends only on k, is still open for both linear and general network codings. Interestingly, there is a graph which admits network coding for the (unique) finite field of size 4 but not for any finite field whose size is not a square number (for example, 5, 7, 8) [17].

Coding operations. Linear coding over a finite-field alphabet is one of the most useful and popular settings. As mentioned before, linear coding is enough for any graph in the multicast model and for many "natural" graphs in other models. However, there exist counter examples: in [19] it is shown that some graph class from [16] and a simple graph by Koetter do not have a linear network coding even if their alphabet sizes are arbitrarily large, but do have vector-linear network coding over an alphabet of size four (actually $\mathbb{F}_2^2$). From simply transforming Koetter's graph to a graph in the k-pair model by the method in [4], it also turns out that this situation, i.e., insufficiency of linear coding, happens as early as $k = 8$. (Recall that linear coding is sufficient when k=2.) Dougherty et al. [3] found an example that does not have a vector linear network coding over any alphabet, but has a network coding over a size-four alphabet if we allow some non-linear operations.

The number of encoding vertices. Recall that this issue is also very important in this paper. Several studies on reducing the number of encoding vertices do exist, but all of them focus on the multicast model. For the multicast model with two inputs (as in Figure 3), Tavory et al. [21] showed that the number of encoding vertices to construct a network coding is independent of the size of the graph, and Fragouli and Soljanin [6] independently proved that it is bounded by the number of the sinks. For the case of three or more inputs, Langberg et al. [14,13] gave an efficient construction of network coding in which the number of encoding vertices is independent of the size of the graph (only dependent on the numbers of inputs and sinks). This might seem to be closely related to our present result but the property specific to the multicast model plays a key role in their proof.

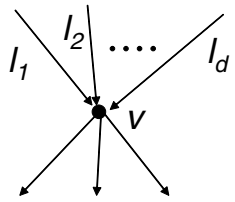

Fig. 4. Vertex v

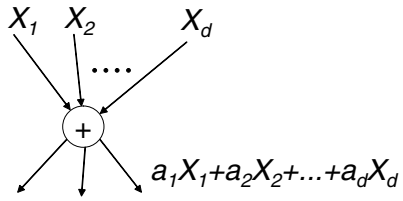

Fig. 5. Linear gate replacing v

2 Network Coding

There are only a finite number of different finite fields if their size is fixed. Therefore we do not lose generality if we fix a finite field itself. For a finite field $\mathbb{F}$, *Linear Network Coding over* $\mathbb{F}$ (LNC($\mathbb{F}$)) is the following decision problem: we are given a directed acyclic graph (DAG) $G = (V, E)$ ($|V| = n$, $|E| = m$) and a *requirement* R. In this paper we do not discuss specific values of exponents for polynomials representing the complexity, and thus we simply use n as the size of the graph though it is $n+m$ precisely. For each vertex $v \in V$, its incoming edges have labels $l_1, l_2, \ldots, l_d$ where d is the indegree of v. R is a set of *source-sink pairs*, given as $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, where, for all i, j, (i) $s_i, t_i \in V$, $s_i \neq s_j$ (for $i \neq j$) and $t_i \neq t_j$ (for $i \neq j$), and (ii) the indegree of s_i and the outdegree of t_i are both zero. Consider a mapping σ, called a *gate assignment mapping*, such that for a vertex v of indegree d, $\sigma(v)$ is given as $\sigma(v) = (a_1, \ldots, a_d)$ where each a_i is in $\mathbb{F}$.

Then σ maps the graph G to a linear circuit in the following sense: suppose that $\sigma(v) = (a_1, \ldots, a_d)$ for a vertex v. Then as shown in Figures 4 and 5, we replace the vertex v by the linear gate computing $a_1X_1 + a_2X_2 + \cdots + a_dX_d$, where X_i is the state (in $\mathbb{F}$) of the incoming edge with label l_i. So, we can consider that a_i is assigned to the edge labeled by l_i. In this paper, we assume that all outgoing edges from a single vertex have the same output state. However, this assumption can be removed by increasing (polynomially) the number of vertices by a method similar to Lemma 1 given later. Thus, this restriction does not lose generality. Source vertices receive global inputs $(x_1, x_2, \ldots, x_k) \in \mathbb{F}^k$ of the circuit and sink vertices hold global outputs $(y_1, \ldots, y_k) \in \mathbb{F}^k$ of the circuit. Note that y_i is a function, denoted by $f_i(x_1, \ldots, x_k)$, depending on the k global input values in general. LNC($\mathbb{F}$) asks whether some gate assignment mapping σ can yield $f_i(x_1, \ldots, x_k) = x_i$ for all $1 \leq i \leq k$. If such σ exists, we say that σ *realizes* the requirement R and also that a *network coding exists* for the graph G and the requirement R. To summarize:

Linear Network Coding over $\mathbb{F}$ (LNC($\mathbb{F}$))
Instance: A pair (G, R) where G is a DAG and R is a set of k source-sink pairs.
Question: Is there a network coding, i.e., is there a gate assignment mapping (over $\mathbb{F}$) which realizes (G, R)?

Assumption: We assume that the number k of source-sink pairs is a constant.

Now we introduce a restriction to the graph which does not lose generality: a DAG G is said to be 2/1-restricted if the (indegree, outdegree) pair for each vertex is only either $(2,1)$ or $(1,2)$, except for sources and sinks.

Lemma 1. *([14]) If LNC($\mathbb{F}$) is solvable for 2/1-restricted graphs in polynomial time, then it is solvable for general graphs in polynomial time.*

Thus, without loss of generality we can assume that our graph contains k $(0,1)$ vertices (sources), k $(1,0)$ vertices (sinks) and all the others are $(2,1)$ or $(1,2)$ vertices. For such a G, a gate assignment mapping is given as $\sigma(v) = (a_1, a_2)$ (where $a_1, a_2 \in \mathbb{F}$) for $(2,1)$ vertices v and $\sigma(v) = a$ (where $a \in \mathbb{F}$) for $(1,2)$ and $(1,0)$ vertices v. Hence the number of different operations at each vertex is at most $|\mathbb{F}|^2$ and so the number of all different mappings is $O(|\mathbb{F}|^{2n})$, which is a trivial upper bound for the complexity of the problem (it is straightforward to check for each assignment whether it actually realizes R).

In the following two sections, we shall explain how to reduce the time complexity to polynomial. In the next section, we first prove that we need only a constant number, C, of "real" $(2,1)$ vertices. If $\sigma(v) = (a_1, a_2)$, "real" means neither a_1 nor a_2 is zero, and we call such a vertex a *coding vertex*. Note that if (at least) one of them is zero, then the corresponding incoming edge can be "cut" and the vertex effectively becomes a $(1,1)$ vertex. Thus the number of different cases to determine the positions of those coding vertices is at most $\binom{n}{C}$, which is bounded by a polynomial. Since the number of different coding operations at each vertex is also constant (at most $|\mathbb{F}|^2$), we can check all the possibilities just by increasing the complexity by a constant factor. For each set of fixed positions and coding operations for those coding vertices, all we have to do further is to decide how to change the other $(2,1)$ vertices to (effective) $(1,1)$ vertices. An exhaustive search would still need exponential time for this purpose, but as shown in Section 4, we can use the disjoint-paths problem for which polynomial-time algorithms are known.

3 Main Lemma

We prove the key lemma which bounds the number of coding vertices.

Lemma 2. *If R is realized by some gate assignment mapping, then it is realized by another mapping for which the number of coding vertices is at most $|\mathbb{F}|^{3k}$.*

Proof. We need several new definitions. Consider the linear circuit T that is induced from the requirement $R = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ and a gate assignment mapping σ realizing R. Let $x_1, \ldots, x_k \in \mathbb{F}$ be the global inputs (at the sources) of T. Then the input and output values of each gate g (which corresponds to a coding vertex) can be written in the form $\sum_{i=1}^{k} a_i x_i$ where $a_i \in \mathbb{F}$. First, we define the following change of functionality of a gate. Let g be a gate which

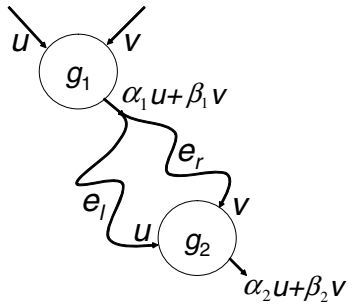

Fig. 6. Gates g_1 and g_2 of the same type

produces output $\alpha X + \beta Y$ for left input X and right input Y. The *left-side γ-change* of g means that we change g to the gate which produces output $(\alpha + \gamma)X + \beta Y$. In particular, if $\gamma = -\alpha$ we call the change the *left-side cut* of g. Similarly we define the *right-side γ-change* of g and the *right-side cut* of g. Note that by the left-side γ-change, the output value O_g of g changes to $O_g + \gamma X$ (that is, changes by $+\gamma X$). Secondly, we define the *effect* of a gate g on the i-th global output y_i (at sink t_i) of T. This is defined as the value of y_i (in $\mathbb{F}$) when all the inputs of T are set to 0 while the output value of g is forced to 1 (regardless of the input of g), or equivalently, is defined as the sum of products of all the σ values assigned to the edges on the paths from g to the i-th global output. Similarly we can define the effect of g on the left input or the right input of another gate. The following lemma follows easily from the linearity of gates.

Lemma 3. *Let $e \in \mathbb{F}$ be the effect of a gate g on the i-th output y_i of T. If the output value of g is changed (due to a change of its functionality) by $+\mu$, then the value of y_i is changed by $+e\mu$. A similar statement holds for the left and the right inputs of a gate, instead of y_i.*

Finally, we define the notion of a type of each gate. Let $a = (a_1, \ldots, a_k)$, $b = (b_1, \ldots, b_k)$ and $e = (e_1, \ldots, e_k)$, where all a_i, b_i, e_i are in $\mathbb{F}$. A gate g is of *type* (a, b, e) if the left-side and right-side inputs of g are $\sum_{i=1}^{k} a_i x_i$ and $\sum_{i=1}^{k} b_i x_i$, respectively, and the effect of g on the j-th output of T is e_j. The following lemma is straightforward from Lemma 3.

Lemma 4. *Fix arbitrary global input values and let g_1 and g_2 be two gates of the same type where g_1 is not a descendant of g_2 (see Figure 6). Suppose that the output value of g_1 is changed (again due to a change of its functionality) by $-\mu$ and suppose that then the output value of g_2 is similarly changed by $+\mu$. Then the outputs of T do not change their values.*

Now suppose that the number of gates in T is larger than $|\mathbb{F}|^{3k}$. Then there are two gates g_1 and g_2 whose types are the same, (a, b, e) say. Let $g_1(X, Y) = \alpha_1 X + \beta_1 Y$ and $g_2(X, Y) = \alpha_2 X + \beta_2 Y$ (without loss of generality, we can assume that none of $\alpha_1, \beta_1, \alpha_2, \beta_2$ are 0). Again assume that g_1 is not a descendant of g_2

(see Figure 6). Then we have the following lemma (Lemma 5), which completes the proof of Lemma 2.

Lemma 5. *Let g_1 and g_2 be as above. Then there must be a left-side cut (and/or a right-side cut) of g_1 and a left-side α_2'-change (and/or a right-side β_2'-change) of g_2 such that the functionality of the circuit does not change.*

Proof. Let e_l and e_r be the effects of g_1 on the left and right inputs of g_2, respectively (see Figure 6). Then, we consider the following three cases: (i) $\alpha_1 e_l \neq 1$ (ii) $\beta_1 e_r \neq 1$ (iii) $\alpha_1 e_l = \beta_1 e_r = 1$. We only analyze cases (i) and (iii) since the analysis of (ii) is similar to (i). Fix arbitrary global input values and let u and v be the values of (both) g_i's left and right inputs, respectively.

Case (i): $\alpha_1 e_l \neq 1$. We show that after the left-side cut of g_1 the output value of g_1 is changed by $-\alpha_1 u$ and there is a choice of α_2' such that the output value of g_2 is changed by $+\alpha_1 u$ after the left-side α_2'-change of g_2. Then, the lemma follows from Lemma 4. In fact, by the left-side cut of g_1 the output value of g_1 is changed from $\alpha_1 u + \beta_1 v$ to $\beta_1 v$. Thus, the output of g_1 is changed by $-\alpha_1 u$. Then, by Lemma 3, the left and right input values of g_2 are changed by $-e_l\alpha_1 u$ and $-e_r\alpha_1 u$, respectively. This means that the left and right inputs of g_2 take the values $u - e_l\alpha_1 u$ and $v - e_r\alpha_1 u$, respectively. (Recall that g_2 is of the same type as g_1, which means that its previous input values were u and v.) By the left-side α_2'-change of g_2, the output value of g_2 then changes by

$$\alpha_2'(u - e_l\alpha_1 u) = \alpha_2'(1 - e_l\alpha_1)u. \tag{1}$$

By setting $\alpha_2' = \alpha_1(1 - e_l\alpha_1)^{-1}$, the right-hand side of Equation (1) is $+\alpha_1 u$. Thus the output value of g_2 changes by $+\alpha_1 u$.

Case (iii): $\alpha_1 e_l = \beta_1 e_r = 1$. First we carry out the left-side and the right-side cuts of g_1. Then, the output value of g_1 is changed by $-(\alpha_1 u + \beta_1 v)$. By Lemma 3 the left and right input values of g_2 are changed by

$$-e_l(\alpha_1 u + \beta_1 v) = -\frac{1}{\alpha_1}(\alpha_1 u + \beta_1 v) = -u - \frac{\beta_1}{\alpha_1} v$$

and

$$-e_r(\alpha_1 u + \beta_1 v) = -\frac{1}{\beta_1}(\alpha_1 u + \beta_1 v) = -v - \frac{\alpha_1}{\beta_1} u,$$

respectively. Thus, the left and right inputs of g_2 take values $u - u - \frac{\beta_1}{\alpha_1} v = -\frac{\beta_1}{\alpha_1} v$ and $v - v - \frac{\alpha_1}{\beta_1} u = -\frac{\alpha_1}{\beta_1} u$, respectively. Then by the left-side α_2'-change and the right-side β_2'-change of g_2, the output value of g_2 changes by

$$-\alpha_2' \frac{\beta_1}{\alpha_1} v - \beta_2' \frac{\alpha_1}{\beta_1} u.$$

By setting $\alpha_2' = -\alpha_1$ and $\beta_2' = -\beta_1$ of g_2, we can verify that the output value of g_2 changes by $+(\alpha_1 u + \beta_1 v)$. Therefore, the lemma follows from Lemma 4 in this case also.

This concludes the proof of Lemma 2. □

4 Polynomial-Time Algorithms

Recall that our graph includes only $(2,1)$ and $(1,2)$ vertices excepting sources (i.e., $(0,1)$ vertices) and sinks (i.e., $(1,0)$ vertices). Fix a gate assignment mapping σ. Then σ defines a linear circuit, T, as described in Section 2. Now we introduce a procedure which simplifies T:

Call an edge *dead* if it is assigned 0 by σ.
(i) Remove all dead edges.
(ii) If all outgoing edges of a vertex v are removed, then remove all its incoming edges, too.
(iii) Repeat (ii) until no further removal is possible.

The resulting circuit is said to be *reduced* and includes the following vertices: (i) k source vertices (all of them should remain if σ realizes the requirement R), (ii) k sink vertices (similarly as above), (iii) vertices having two remaining incoming edges, called *coding vertices*, (iv) vertices having two remaining outgoing edges, called *fork vertices*, and (v) all other vertices, called *path vertices*, such that for each of them there remains a single incoming and a single outgoing edge. We say that a mapping σ is *reduced* if it directly defines a reduced circuit (i.e., if 0-assigned edges are removed, then no further simplification is possible). Proofs for the following two lemmas are straightforward and omitted.

Lemma 6. *Suppose that there exists a gate assignment mapping σ which realizes a requirement R. Then there exists a reduced mapping σ' such that: (i) σ' realizes R; and (ii) the number of coding vertices of σ' is less than or equal to the number of coding vertices of σ.*

Lemma 7. *If σ is reduced, then the circuit defined by σ has the same numbers of coding vertices and fork vertices.*

Now we are ready to prove our main result.

Theorem 1. *LNC($\mathbb{F}$) can be solved in polynomial time.*

Proof. For a given (G,R), suppose that there is a gate assignment mapping which realizes R. Then, by Lemma 2, there is such a mapping σ which includes only a finite number C of coding vertices. This implies, by Lemma 6, that there is a reduced mapping σ' which realizes R and includes at most C coding vertices. Lemma 7 guarantees that the circuit defined by σ' includes at most C fork vertices whose two outgoing edges are assigned non-zero values by σ'. Thus to prove the existence of σ, it suffices to find a reduced mapping σ' which (i) includes at most C coding vertices and at most C fork vertices and (ii) realizes R. For (i), our basic strategy is enumeration in polynomial time of all such mappings without considering the requirement R. Checking (ii) can obviously be done in polynomial time.

For the enumeration, we first select from V a set of (at most) C positions (vertices) for C coding vertices and (at most) C positions for C fork vertices.

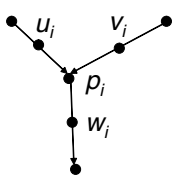

Fig. 7. Inserting u_i, v_i, w_i at a code vertex

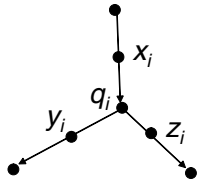

Fig. 8. Inserting x_i, y_i, z_i at a fork vertex

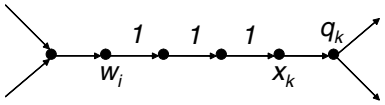

Fig. 9. Assigning 1 to all vertices on the path

Note that the total number of such selections is $O(\binom{n}{C} \cdot \binom{n}{C}) = O(n^{2C})$, and is bounded by a polynomial since C is constant. Suppose that a single selection consists of $(2,1)$ vertices $p_1, \ldots, p_C$ for the coding vertices and $(1,2)$ vertices $q_1, \ldots, q_C$ for the fork vertices. Then we make another change to the graph G as follows: insert vertices u_i, v_i, w_i into the three edges of p_i and x_i, y_i, z_i into the three edges of q_i as shown in Figures 7 and 8. After this modification, let

$$\text{OUT} = \{s_1, \ldots, s_k, w_1, \ldots, w_C, y_1, \ldots, y_C, z_1, \ldots, z_C\}$$
$$\text{IN} = \{t_1, \ldots, t_k, u_1, \ldots, u_C, v_1, \ldots, v_C, x_1, \ldots, x_C\}.$$

The key observation is that a sequence of vertices of path vertices in (v) defined by σ' constitutes a "path" in graph G which starts from some vertex in OUT and ends with some vertex in IN. Furthermore, any two such paths are obviously vertex-disjoint, i.e., these paths define a matching between OUT and IN. Conversely, suppose that we are given fixed OUT and IN (let $|\text{OUT}| = |\text{IN}| = L$) and a (complete) matching $M = \{(a_1, b_1), (a_2, b_2), \ldots, (a_L, b_L)\} \subseteq \text{OUT} \times \text{IN}$. Then this matching (together with the set of fork vertices and a concrete operation at each of the coding vertices) defines a linear circuit, and whether or not such a circuit is defined by some σ' is equivalent to whether or not there are vertex-disjoint paths connecting each a_j to b_j, without going through any p_i or q_i (where $1 \leq i \leq C$). If L is constant, then this test can be done in time polynomial in n [5]. If such paths exist, then we can construct the circuit and the corresponding gate assignment mapping σ'. It is straightforward to check whether σ' realizes the requirement R of network coding. Note that without loss of generality we can assign 1 to the vertices on the path as shown in Figure 9, since all the multiplicative constants on the path can be accumulated into the constant on edge (x_k, q_k). Algorithm 1 shows the formal description of our algorithm.

Since C is constant, the number of repetitions of Line 2 is bounded by a polynomial. For fixed CODE and FORK, the number of repetitions of Line 4

Algorithm 1. Our Algorithm for Linear Network Coding of k Source-sink Pairs.

```
1: Compute the maximum number C of coding vertices by Lemma 2.
2: for each CODE ⊆ V, FORK ⊆ V such that |CODE| = |FORK| ≤ C do
3:    Insert vertices as in Figures 7 and 8 for vertices in CODE and FORK and compute OUT and IN.
4:    for each matching M ⊆ OUT × IN do
5:       Check if there are disjoint paths for M in G' where G' is the graph obtained
         by removing all vertices (and their incoming and outgoing edges) in CODE ∪
         FORK (but vertices in IN and OUT and all vertices in V − CODE ∪ FORK).
6:       if No then
7:          Go to end.
8:       else
9:          for each gate assignment mapping σ for G' such that σ assigns (i) 1 to all
            the edges in the selected paths above, (ii) any a ∈ F − {0} to each of the
            edges to or from vertices in CODE∪FORK, and (iii) 0 to all the other edges
            do
10:            Check if σ realizes R.
11:            if Yes then
12:               Answer YES and halt.
13:            end if
14:         end for
15:      end if
16:   end for
17: end for
18: Answer NO and halt.
```

is bounded by a constant. The number of repetitions of Line 9 is also bounded by a constant. Line 5 can be done in polynomial time by [5], and all the other instructions can obviously be executed in polynomial time. Thus the total running time is also polynomial. The correctness of the algorithm follows from the observation given in the previous pages.

5 Concluding Remarks

Our algorithm runs in polynomial time. However, its exponent is obviously not small. Seeking more efficient algorithms is an obvious future task. Fixed parameter tractability of the problem also seems interesting, but it is known that the disjoint paths problem for DAGs is $W[1]$-hard [20]. Therefore, in order to design FPT algorithms, we have to bypass the main subroutine solving this problem, which does not seem easy.

References

1. Adler, M., Harvey, N.J., Jain, K., Kleinberg, R.D., Lehman, A.R.: On the capacity of information networks. In: Proc. 17th ACM-SIAM SODA, pp. 241–250 (2006)
2. Ahlswede, R., Cai, N., Li, S.-Y.R., Yeung, R.W.: Network information flow. IEEE Transactions on Information Theory 46, 1204–1216 (2000)

3. Dougherty, R., Freiling, C., Zeger, K.: Insufficiency of linear coding in network information flow. IEEE Transactions on Information Theory 51, 2745–2759 (2005)
4. Dougherty, R., Zeger, K.: Nonreversibility and equivalent constructions of multiple-unicast networks. IEEE Transactions on Information Theory 52, 5067–5077 (2006)
5. Fortune, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. Theoret. Comput. Sci. 10, 111–121 (1980)
6. Fragouli, C., Soljanin, E.: Information flow decomposition for network coding. IEEE Transactions on Information Theory 52, 829–848 (2006)
7. Harvey, N.J., Karger, D.R., Murota, K.: Deterministic network coding by matrix completion. In: Proc. 16th ACM-SIAM SODA, pp. 489–498 (2005)
8. Harvey, N.J., Kleinberg, R.D., Lehman, A.R.: Comparing network coding with multicommodity flow for the k-pairs communication problem. MIT LCS Technical Report 964 (September 2004)
9. Ho, T., Karger, D.R., Médard, M., Koetter, R.: Network coding from a network flow perspective. In: Proc. IEEE International Symposium on Information Theory (2003)
10. Jaggi, S., Sanders, P., Chou, P.A., Effros, M., Egner, S., Jain, K., Tolhuizen, L.M.G.M.: Polynomial time algorithms for multicast network code construction. IEEE Transactions on Information Theory 51, 1973–1982 (2005)
11. Koetter, R.: Network coding home page, `http://tesla.csl.uiuc.edu/~koetter/NWC/`
12. Koetter, R., Médard, M.: Beyond routing: An algebraic approach to network coding. In: Proc. 21st Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 122–130 (2002)
13. Langberg, M., Sprintson, A., Bruck, J.: Network coding: A computational perspective. In: Proc. 40th Conference on Information Sciences and Systems (2006)
14. Langberg, M., Sprintson, A., Bruck, J.: The encoding complexity of network coding. IEEE Transactions on Information Theory 52, 2386–2397 (2006)
15. Lehman, A.R.: Network Coding. PhD thesis. MIT, Cambridge (2005)
16. Lehman, A.R., Lehman, E.: Complexity classification of network information flow problems. In: Proc. 15th ACM-SIAM SODA, pp. 142–150 (2004)
17. Lehman, A.R., Lehman, E.: Network coding: Does the model need tuning? In: Proc. 16th ACM-SIAM SODA, pp. 499–504 (2005)
18. Li, S.-Y.R., Yeung, R.W., Cai, N.: Linear network coding. IEEE Transactions on Information Theory 49, 371–381 (2003)
19. Médard, M., Effros, M., Ho, T., Karger, D.: On coding for non-multicast networks. In: Proc. 41st Annual Allerton Conference on Communication, Control and Computing (2003)
20. Slivkins, A.: Parameterized tractability of edge-disjoint paths on directed acyclic graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 482–493. Springer, Heidelberg (2003)
21. Tavory, A., Feder, M., Ron, D.: Bounds on linear codes for network multicast. ECCC Technical Report 33 (2003)
22. Wang, C.-C., Shroff, N.B.: Beyond the butterfly – A graph-theoretic characterization of the feasibility of network coding with two simple unicast sessions. In: Proc. IEEE International Symposium on Information Theory (2007)
23. Wang, C.-C., Shroff, N.B.: Intersession network coding for two simple multicast sessions. In: Proc. 45th Annual Allerton Conference on Communication, Control and Computing (2007)