# Network Coding is Highly Non-Approximable

Hongyi Yao
Institute of Theoretical Computer Science
Tsinghua University
Beijing, 100084, P.R China
Email: yaohongyi03@gmail.com

Elad Verbin
Institute of Theoretical Computer Science
Tsinghua University
Beijing, 100084, P.R China
Email: elad.verbin@gmail.com

***Abstract*—We address the network coding problem, in which messages available to a set of sources must be passed through a network to a set of sinks with specified demands.**

**It is known that the problem of deciding whether the demands can be met using a linear code is NP-hard [Lehman and Lehman, SODA '04]. Such a result is not known if we allow general (=non-linear) codes to be used. Despite this, network coding is believed to be a very hard problem both when restricting to linear codes, and when considering general codes.**

**In the current paper we give some evidence for this hardness. Call a sink *happy* if it receives all of the data it demands. We show that the problem of maximizing the number of happy sinks by a general network code is NP-hard to approximate to within a multiplicative factor of $n^{1-\epsilon}$, for any $\epsilon > 0$. Here, $n$ is the number of sinks. To our knowledge, this is the first hardness result known for general network coding. The same holds for maximizing the number of happy sources.**

**Let $n_s$ be the number of sinks. We also prove a stronger result about linear codes: that given a network that can be satisfied by a linear code, it is NP-hard to find a linear code that makes at least $22$ sinks happy. In particular, this means that the problem of maximizing the number of happy sinks in a linear code cannot be approximated to any factor better than $n_s/22$, even for arbitrarily large $n_s$ – this is the harshest kind of inapproximability possible.**

## I. Introduction

Network coding can be seen as a variant of multicommodity flow, where commodities can be duplicated or combined (using computational operations) at any node. The network (represented by a directed graph) consists of multiple *nodes*, which correspond to *computers*, and *edges*, which correspond to *channels*. A set of *source nodes* has a set of *data items*, and it wants to transmit them to a set of *sink nodes* through the edges. Indeed, coding really helps: in the famous *butterfly network* the approach based on multicommodity flow cannot satisfy all of the sinks, but a *network coding* that satisfies all sinks does exist. See Section II for a formal definition of the network coding problem. A good source on network coding are the books of Yeung [12], [13].

An important class of communication protocols are linear protocols: we think of each data item and each message as an element in a finite field $\mathbb{F}_q$. Each transmitted message is a linear combination of data items. We call a network *satisfiable* if the demands of all sinks can be met. We call it linearly-satisfiable if they can be met by a linear code.

In the case where all sinks want all data items, the problem was solved by [1], [6], [3], who showed that if the network is satisfiable, then a randomly-chosen linear code satisfies it with good probability. However, the more common case where sinks have different demands is wide open, and is believed to be much harder [12], [8]. This is the case we deal with in this paper.

The first paper to consider the computational complexity of network coding is [8]. In it, Lehman and Lehman prove that it is NP-Hard to determine whether a network coding instance is linearly satisfiable. They prove this by a reduction from the 3-SAT problem. For more information about NP-hardness, see Section II-B. [8] only prove that it is NP-hard to determine whether there is a linear protocol that satisfies the network. However, the problem of checking whether a network is (nonlinearly-) satisfiable is not even known to be NP-hard. In this paper, we offer the first NP-hardness proof related to non-linear network coding. Non-linear codes can be much stronger than linear codes, as shown by [5]. Another interesting recent paper is Langberg and Sprintson's [7], who prove that it is hard to approximate the *rate* of a network when using a vector-linear code.[1]

### A. Our Contributions

In a network coding protocol, we call a sink *happy* if it receives all data items that it desires. We call a source $s$ *happy* if all data items contained in $s$ are successfully transmitted to all sinks that want them. It is interesting to try to maximize the number of happy sinks or sources, since even if a network is not satisfiable, it might be interesting to meet at least some of the demands.

It is perfectly plausible that, say, whenever the network is satisfiable, then we can find in polynomial time a protocol where $99\%$ of the sinks are happy; such a situation would mean that the network coding problem has quite good algorithms, that the community might be satisfied with. In this paper, we are sad to dispel such hopes: we prove that problems

[1]Specifically, Langberg and Sprintson proved that the rate is hard to approximate up to a multiplicative constant. There are a few caveats to this: $(i)$ the result assumes the conjecture that it is NP-hard to find a coloring of a 3-colorable graph by $O(1)$ colors (this conjecture is plausible: it is true if the Unique Games Conjecture is true); $(ii)$ The proof works in the non-asymptotic case, i.e. the proof only works when fixing $q$ to a constant rather than letting it tend to infinity; problems that are hard in this non-asymptotic setting are not always hard in the asymptotic setting, since the asymptotic setting is less "jagged".

of this sort are incredibly hard to approximate. In particular, we prove that:[2]

- If we are promised that the network is linearly-satisfiable, then it is NP-hard to find a linear protocol that makes even 22 sinks happy, and the same holds when considering sources instead of sinks. See Section IV.
- Let $n_s$ be the number of sinks. Considering general (=non-linear) codes, we prove that for any $\epsilon > 0$, it is NP-hard to distinguish between the case that $n_s^{1-\epsilon}$ sinks can be made happy, and the case that no $n_s^{\epsilon}$ sinks can be made happy. The same holds when considering sources instead of sinks. To our knowledge, this is the first hardness result known for general network coding. See Section III.

Two important comments: Firstly, all of our results hold even when the number of sources and sinks tend to infinity. This implies that the problem of maximizing the number of happy sinks in a linear code cannot be approximated to any factor better than $n_s/22$, even for arbitrarily large $n_s$ – this is the harshest kind of inapproximability possible. Secondly, all of our results can be adapted to hold even for the case that each source contains only one unique data item. This might be important for comparing our results with previous results.

The first set of results, that deal with linear protocols, are proved by reductions from variants of the 3-SAT problem. The latter result that deals with non-linear protocols, is a reduction from the maximum independent set problem.

Note that the last result does not imply that it is NP-hard to determine whether a network coding instance is satisfiable. This is still unknown, and is a major open problem.

## II. Preliminaries

We start with some definitions in graph theory. For basic definitions, see [4]. In a directed graph let $deg^-(v)$ and $deg^+(v)$ denote the in-degree and out-degree of $v$, respectively. For an undirected graph, we denote $v \sim_G u$ if $u$ and $v$ are neighbors. For a vertex $v$ and edge $e$, we write $v \in e$ if $v$ is one of the sides of $e$.

### A. Network Coding

The formal definitions related to network coding can be found in [12], [13]. Here we give a short review. In the decision version of a network coding instance $\mathcal{N}(G, S, R, D, \{Items_s\}_{s\in S}, \{Demands_r\}_{r\in R})$, there is a directed acyclic graph $G = (V, E)$ in which parallel edges are allowed, a set of source nodes $S \subseteq V$ with no in-edges, a set of sink (or *receiver*) nodes $R \subseteq V$, a set of data items $D$, a family of item sets $\{Items_s\}$ for each source node, where $Items_s \subseteq D$, and a family of demand sets $\{Demands_r\}$ for each sink node, where $Demands_r \subseteq D$. Fix a number $q$, which is the *alphabet* size, and let the alphabet be $\Sigma = \{1, 2, \ldots, q\}$. For a particular $q$, a *protocol* for this network is a family of mappings. For each edge $e = (u, v)$, the mapping is a function $f_e : \Sigma^{|deg^-(u)|} \to \Sigma$. If $u$ is a source node, then $f_e : \Sigma^{|Items_u|} \to \Sigma$. The mapping $f_e$ specifies how the symbols received by $u$ are translated to the symbol that is sent from $u$ to $v$.

A protocol is said to be *linear* if $\Sigma$ is a finite field and all $f$'s are linear transformations. A protocol is said to be *vector-linear* if $\Sigma$ can be represented as a cartesian product of finite fields, and $f$ is linear in each coordinate. (We hardly use this definition; see more in [13]). For a particular protocol, we denote the value of the data item stored at a source $w$ by $val(w)$ (assuming there is just one item; we won't use this notation in any other case), and we also denote the value transmitted over a channel $(w, w')$ by $val(w, w')$.

Let $r$ be a receiver. If there is a function that maps the symbols received by $r$ to the values of the items $Demands_r$, then $r$ is said to be *happy*. If a protocol makes all receivers happy, then we say that the protocol *satisfies* the network. If there exists some $q$ such that there exists a protocol that satisfies the network, then we say that the network is *satisfiable*. Note that this definition is *asymptotic*.[3] If this holds for a linear protocol, then we say that the network is *linearly satisfiable*.

When a protocol does not perform any coding operations and just copies items, then we call it a *copying-only* protocol. We'll refer to this concept in Section V-A.

### B. NP-Hardness and Computational Complexity

We give some basic information about NP-hardness. For more, see e.g., [10], [11].

NP-hard is the class of problems which are at least as hard to solve, in the worst case, as any other problem in NP (up to polynomial-time reductions). Many problems are known to be NP-hard. It is widely believed that no NP-hard problem has a polynomial-time algorithm. Remarkably, if a polynomial-time algorithm is found for some NP-hard problem, then all problems in NP are also solvable in polynomial time. This includes difficult problems such as SAT, INDEPENDENT SET, etc. . The standard way to prove that a problem is NP-hard is to reduce to it from some problem which is known to be NP-hard. For example, if we want to reduce from problem $P$ to problem $Q$, then we assume that we have an oracle that solves $Q$ in polynomial time, and we prove that we can solve $P$ in polynomial time. It is important to note that NP-hardness only refers to the worst-case complexity of a problem. That is, NP-hard problems might still have efficient algorithms that work correctly on many inputs.

[2]Consider some maximization problem. Consider the following three types of statements: $(i)$ it is NP-hard to distinguish between the case that the value is $\leq \alpha$ and the case that the value is $\geq \beta$; $(ii)$ given an input with value $\geq \beta$, it is NP-hard to find a solution with value $> \alpha$; $(iii)$ the value cannot be approximated to within a multiplicative factor of $\beta/\alpha$. It is easy to see that $(i)$ implies both $(ii)$ and $(iii)$. Also, for most problems, $(ii)$ implies $(i)$ by self-reducibility. In various places in the paper we phrase our results in all three forms, but all of our results are actually of the strongest form, $(i)$.

[3]It is asymptotic in the sense that we only care about whether there exists such $q$, and $q$ might be very large and may depend on the network itself. This corresponds to caring only about what happens after asymptotically-much "communication rounds". The standard definition of network coding, see e.g. Yeung [12] is slightly different to the one we give, but almost equivalent. Ours is a little simpler to describe. In any case, all of our proofs work for the standard definition as well.

The 3-SAT problem is a well-known NP-hard problem. The input consists of a set $\{x_1, \ldots, x_n\}$ of *variables*, and a set $C_1, C_2, \ldots, C_m$ of *clauses*. Every clause consists of exactly three *literals*, where each *literal* is either a variable or its negation. For example, $\overline{x_3} \vee x_5 \vee \overline{x_9}$ is a possible clause. The instance is *satisfiable* if there is a truth assignment to the variables so that each clause has at least one true literal. We find it useful to consider a restriction of the 3SAT problem, denoted 3SAT-5, where each variable must appears in exactly five clauses and a variable does not appear in any clause more than once. This problem is also NP-hard [10].

Another problem of interest is the maximum independent set problem. An independent set is a set of vertices in a graph no two of which are adjacent. The computational hardness result for maximum independent set is:

*Theorem 1 ([2]):* Given an input graph $G$, the problem of finding the largest independent set in the graph is NP-hard to approximate to within $n^{1-\epsilon}$, for any $\epsilon > 0$.

## III. Hardness Result For General Network Coding

In this section, we consider the general (non-linear) network coding problem. We prove inapproximability of the problems of maximizing the number of happy sources, and of maximizing the number of happy sinks. To our knowledge, no previous works have shown computational hardness for any variant of general (=non-linear) network coding. We prove hardness by showing a reduction from the maximum independent set problem.

We first consider the problem of maximizing the number of happy sources. The following lemma gives the reduction:

*Lemma 2:* There exists a mapping $\Phi$ from graphs to instances of network coding, such that a graph $G = (V, E)$ has an independent set of cardinality $k$ if and only if the network coding instance $\Phi(G)$ has a protocol that makes $k$ receivers happy.

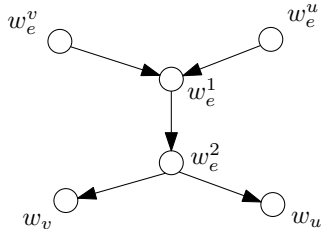


Fig. 1. Gadget for Lemma 2 reduction.

*Proof:* We show the reduction in figure 1. The network $\Phi(G)$ has $|V| + 4|E|$ nodes and $5|E|$ channels. $2|E|$ of the nodes are sources, and $|V|$ are sinks. For each vertex $v$ in $G$, $\Phi(G)$ has a node $w_v$. For each edge $e = (u, v)$ in $G$, $\Phi(G)$ has four nodes: $w_e^v, w_e^u, w_e^1$, and $w_e^2$. $w_e^v$ and $w_e^u$ are sources, and $w_e^1$ and $w_e^2$ are neither sources nor sinks. For each edge $e = (u, v)$ of $G$, the network $\Phi(G)$ contains the following five channels: $(w_e^v, w_e^1), (w_e^u, w_e^1), (w_e^1, w_e^2), (w_e^2, w_v)$, and $(w_e^2, w_u)$. All channels are unit channels, i.e., they are able to transmit one element of the alphabet. Each source node contains exactly one unique data item. The demands $\{Demands_r\}$ are as follows: Each sink $w_v$ demands $deg_G(v)$ data items. Specifically, for each edge $e = (u, v)$, $w_v$ demands the data stored at node $w_e^v$, and $w_u$ wants to get the data stored at node $w_e^u$. This concludes the description of $\Phi(G)$.

We make the following claim:

*Claim 3:* The sink $w_v$ can be made happy if and only if we don't make any of the sinks $\{w_u : (u, v) \in E\}$ happy.

This claim means that we can make a set of sinks happy if and only if it is an independent set of $G$, The lemma clearly follows from this claim. We now prove this claim:

Let $v$ be a vertex of $G$, and let $w_v$ be the corresponding sink. Fix some edge $e = (u, v) \in E$. First observe that in order to make the sink $w_v$ happy, it is necessary that $w_v$ gets the data item $val(w_e^v)$. In order for $w_v$ to get $val(w_e^v)$, it is required that the channel $(w_e^1, w_e^2)$ carries full information about $val(w_e^v)$: this is because there is only one path in the network connecting $w_e^v$ to $w_v$, and that path includes the channel $(w_e^1, w_e^2)$. Furthermore, if $w_v$ is happy, that means that $w_v$ is able to decode the $deg_G(v)$ information items that it needs; since only $deg_G(v)$ channels go into $w_v$, then by a basic information-theoretic argument we get that these channels cannot carry information about any data items except those in $\{val(w_e^v) : u \sim_G v\}$. It follows that if $w_v$ is happy, then $val(w_e^u)$ has *no influence* on $val(w_e^1, w_e^2)$, and in particular $w_u$ *cannot be happy*. We thus see that if $w_v$ is happy, then no node in the set $\{w_u : u \sim_G v\}$ can be happy, and thus the set of happy nodes has to be an independent set.

Conversely, it is easy to see that for any independent set $S \subseteq V$ there is a protocol that makes all vertices $\{w_v : v \in S\}$ happy, by going over every edge $e = (v, u) \in E$ where $v \in S$, and transmitting the value $val(w_e^v)$ from $w_e^v$ to $w_e^1$ to $w_e^2$ to $w_v$. This finishes the proof of the claim, and also the proof of the lemma. ∎

A direct consequence of this lemma is:

*Theorem 4:* In the general network coding problem it is NP-hard to approximate to within $n^{1-\epsilon}$ the number of sinks that can be made happy, for any $\epsilon > 0$.

For maximizing the number of happy sources, we have a similar result, with a very similar proof.

## IV. Hardness Results for Linear Network Coding

We prove hardness by reducing from 3SAT-5. Part of the the reduction in depicted in figure 2. We now describe the reduction. Let $\psi$ be a 3CNF-5 formula with $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$. In the reduction we transform $\psi$ to a network coding problem $\Phi(\psi)$. The network $\Phi(\psi)$ has $4n + 3m$ nodes and $3n + 6m$ channels; $2n$ of the nodes are sources, and $m$ are sinks. The nodes are $\{v_i^T, v_i^F, v_i^1, v_i^2 : 1 \le i \le n\} \cup \{w_j, w_j^1, w_j^2 : 1 \le j \le m\}$. The sources are $\{v_i^T, v_i^F : 1 \le i \le n\}$ and the sinks are $\{w_j : 1 \le j \le m\}$. The channels are: $\{(v_i^T, v_i^1), (v_i^F, v_i^1), (v_i^1, v_i^2) : 1 \le i \le n\} \cup \{(w_j^1, w_j), (w_j^2, w_j), (w_j^2, w_j) : 1 \le j \le m\} \cup \{(v_i^2, w_j^1), (v_i^T, w_j^2) : x_i \in C_j\} \cup \{(v_i^2, w_j^1), (v_i^F, w_j^2) : \overline{x_i} \in C_j\}$. Here, $(w_j^2, w_j)$ appears twice because we put two parallel channels there; also, the notation $x_i \in C_j$ is used to denote that $x_i$ appears in $C_j$ un-negated, and $\overline{x_i} \in C_j$

means that $\overline{x_i}$ appears in $C_j$ (negated). Each channel has unit capacity. Each source contains exactly one unique data item. The demands are as follows: For each clause $C_j$, the sink $w_j$ demands three data items. Let us specify using an example: if the clause is $\overline{x_3} \vee x_5 \vee \overline{x_9}$, then $w_j$ demands the data items from sources $v_3^F$, $v_5^T$, and $v_9^F$. The general rule is clear.

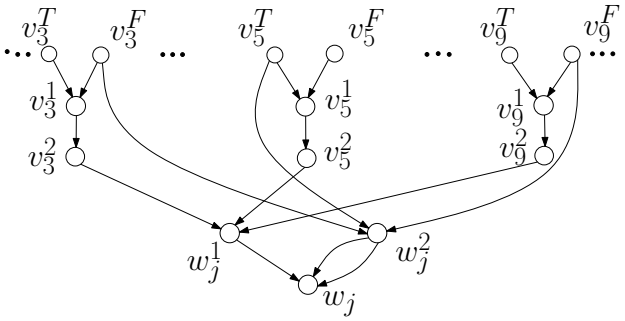


Fig. 2. Gadget for Claim 5 reduction.

We make the following claim:

*Claim 5:* For any set $\{w_j : j \in J\}$ of sinks, there is a linear protocol that satisfies all of these sinks if and only if there is an assignment to $\psi$ that satisfies all of the clauses $\{C_j : j \in J\}$.

*Proof:* Assume we have an assignment $\sigma$ that satisfies all of the clauses $\{C_j : j \in J\}$. Then the following protocol makes all of the sinks in $\{w_j : j \in J\}$ happy: If $\sigma$ sets $x_i$ to TRUE, then copy the data item from $v_i^T$ to $v_i^1$, and then to $v_i^2$, and then to all nodes in $\{w_j^1 : x_i \in C_j\}$, and then to all nodes in $\{w_j : x_i \in C_j\}$. Similarly, if $\sigma$ sets $x_i$ to FALSE, then copy the data item from $v_i^F$ to $v_i^1$, and then to $v_i^2$, and then to all nodes in $\{w_j^1 : \overline{x_i} \in C_j\}$, and then to all nodes in $\{w_j : \overline{x_i} \in C_j\}$.

The data items that $w_j^1$ wants to send to $w_j$ correspond to the literals that cause $C_j$ to be satisfied by $\sigma$. Thus, if $w_j^1$ wants to send more than one data item to $w_j$, then it just arbitrarily picks one of these data items and sends it to $w_j$, and ignores the others.

It is easy to see that by now, each of the sinks in $\{w_j : j \in J\}$ receives at least one of the data items that it wants. It can get the two others through $w_j^2$, using the double channel. Thus we see that all of $\{w_j : j \in J\}$ can be made happy. It is interesting to see that this protocol only copies data items, i.e., it is a copying-only protocol. We shall return to this point in Section V-A.

Let us now prove the converse: that if there is a linear protocol that satisfies all sinks in $\{w_j : j \in J\}$ then we can build an assignment $\sigma$ that satisfies all clauses $\{C_j : j \in J\}$. The assignment $\sigma$ assigns to $x_i$ the value TRUE if $val(v_i^1, v_i^2)$ depends only on $val(v_i^T)$, and assigns the value FALSE if $val(v_i^1, v_i^2)$ depends only on $val(v_i^F)$. (When we say that $val(w_1, w_2)$ "depends only on" $val(w)$ we mean that it does not depend on data items stored in any source except $w$). In any other case, i.e. if $val(v_i^1, v_i^2)$ depends on both or none of the values $val(v_i^T), val(v_i^F)$, then $\sigma$ assigns to $x_i$ an arbitrary value. We proceed to prove that for each $j \in J$, this assignment indeed satisfies $C_j$.

Fix some happy sink $w_j$ where $j \in J$. Let's assume for example that the clause $C_j$ is $\overline{x_3} \vee x_5 \vee \overline{x_9}$ (the proof works the same way in general). Denote by $D_j = \{val(v_3^F), val(v_5^T), val(v_9^F)\}$ the set of data items that $w_j$ wants (and gets, since $w_j$ is happy), and denote by $D_j^c$ all other data items. Since the code is linear, then every message that $w_j$ receives is a linear combination of data items. We claim that each message that $w_j$ receives does not depend on any data item in $D_j^c$. Assume in contradiction that $w_j$ does receive some message that depends on items in $D_j^c$. Then, since $w_j$ also knows the three values in $D_j$, then it can use linear algebra to "cancel" $D_j$ out from that message, and then $w_j$ knows some linear combination of the items in $D_j^c$; this is not possible, since it means that $w_j$ gets four independent pieces of information, while $w_j$ only has three ingoing channels. Thus, each message that $w_j$ gets only depends on the items in $D_j$.

Furthermore, since $w_j$ has three ingoing channels and needs three pieces of information, it is clear that the message $val((w_j^1, w_j))$ must carry some information about the items in $D_j$. Suppose w.l.o.g. that $val((w_j^1, w_j))$ depends on $val(v_3^F)$. Observe that if the message transmitted over $(v_3^1, v_3^2)$ depends on $val(v_3^T)$, then since the code is linear and there is no other channel through which to get $val(v_3^T)$, it means that this dependence must carry on, and then $val((w_j^1, w_j))$ will depend on $val(v_3^T)$, which is a contradiction. This means that the message transmitted over $(v_3^1, v_3^2)$ cannot depend on $val(v_3^T)$ and will depend only on $val(v_3^F)$, and by the definition of $\sigma$, $\sigma(x_3) = FALSE$, so $C_j$ will be satisfied. This concludes the proof of the lemma. ■

We now somewhat alter the reduction.

*Lemma 6:* The reduction can be changed so that the network only has 22 sources.

*Proof:* To make the number of sources smaller, we change the reduction of Claim 5 by *unifying* some of the sources. Suppose the two literals $x_3$ and $x_7$ (for example) never appear together in the same clause. Then we claim that the sources $v_3^T$ and $v_7^T$ can be unified, i.e., replaced by one source that contains both of the data items, without hurting the correctness of the reduction. This is easy to check by just going over the proof of Claim 5, and seeing that the proof only requires the assumption that $val(v_3^1, v_3^2)$ does not depend on any of the values at $v_5^T, v_5^F, v_9^T, v_9^F$, and similarly with $val(v_5^1, v_5^2)$ and $val(v_9^1, v_9^2)$. (Using the clause $C_j = \overline{x_3} \vee x_5 \vee \overline{x_9}$ as an example). We also must never unify the node of a literal with the node of its negation.

It's easy to see that unifying the nodes of two literals that do not appear together in any clause keeps the assumption true.

We now show that we can unify sources in this way, such that in the end we are only left with 22 sources. To see this, suppose that we can color the variables of $\psi$ by $k$ colors, such that no two variables with the same color co-appear in the same clause. In this case, we can unify source nodes to make the number of sources $2k$, by unifying the nodes $\{v_i^T\}$ for each color class, and the nodes $\{v_i^F\}$ for each color class. Thus, it suffices to prove that we can color the variables of $\psi$

by $k = 11$ colors. To see this, write a graph $H_\psi$ where the vertices are the variables of $\psi$ and there is an edge between two variables if they co-appear in some clause. We want to color this graph by 11 colors such that no edge is monochromatic. Since $\psi$ is a 3CNF-5 formula, then each variable co-appears with at most 10 other variables. Thus, the maximum degree in $H$ is at most 10, and we can use the greedy algorithm to color $H$ in $k = 11$ colors.[4] ∎

We can in fact get the number of sources down from 22 to 10 by reducing from PLANAR-3SAT. The reduction uses the fact that planar graphs can be colored by 5 colors in polynomial time. The details are omitted. It seems interesting whether 2 sources suffice.

We now give our main theorem.

*Theorem 7:* In the problem of maximizing the number of happy sources using a linear network code, it is NP-hard to distinguish between the case that all sources can be made happy, and the case that at most 21 sources can be made happy. This holds even if the number of sources, $n_s$, is arbitrarily large.

*Proof:* Let $n_s$ be the number of sources (arbitrarily large). We again reduce to 3SAT-5. Let $\psi$ be a 3CNF-5 formula. We transform it to a network coding instance $\mathcal{N}$ as follows: The network $\mathcal{N}$ consists of $n_s$ sources. For each set of 22 out of the $n_s$ sources, we place an instance produced by the reduction described in Lemma 6 over these 22 sources. Thus, we perform the reduction $\binom{n_s}{22}$ times, each time producing fresh and distinct data items, distinct channels, distinct internal nodes and distinct sinks. The only thing in common is the sources. At the end we get a network $\mathcal{N}$ with $\binom{n_s}{22} \cdot (2n + 3m) + n_s$ nodes and $\binom{n_s}{22} \cdot (3n + 6m)$ channels. It is easy to see that if $\psi$ is satisfiable, then all sources of $\mathcal{N}$ can be made happy, and that if $\psi$ is not satisfiable, then no set of 22 sources of $\mathcal{N}$ can be made happy. Thus, if we can distinguish between the two cases, then we have a solution to 3SAT-5. It follows that it is NP-hard hard to distinguish between the case that all sources can be made happy, and the case that at most 21 sources can be made happy. ∎

## V. Discussion and Open Questions

### A. A Note About Our Hardness Reductions

It is interesting to observe that in the reduction of Claim 5, if the network is linearly satisfiable then it can be satisfied by a copying-only protocol. The paper [9] presents an algorithm which in polynomial-time finds a vector-linear protocol that satisfies the network if there exists a copying-only protocol that satisfies the network. Thus, our reduction cannot prove an analogue of Theorem 7 for vector-linear protocols.

[4]The greedy algorithm works as follows: go over the vertices one by one is some arbitrary order, and color each vertex $v$ by some color not used by its already-colored neighbors. Since there are 11 colors but only at most 10 neighbors, then there must be some such free color, so this process produces a valid coloring. Also note that this process runs in linear time. This is important since the coloring needs to be produced as part of the reduction, and reductions need to run in polynomial time.

Thus, it seems we might have gotten close to the limits of what can be proved using gadgets that admit copying-only solutions. The study of gadgets that require coding operations to satisfy tends to be much more complicated, as demonstrated by the many caveats in [7]. It seems that we are missing techniques that can deal with complicated networks that require not-only-copying protocols.

### B. Open Questions

- It is still open to determine to complexity of checking whether a network is satisfiable. Is it NP-hard? Is it decidable?
- It seems interesting to prove that checking whether a network is linearly satisfiable is NP-hard even for networks where the number of sources, sinks, and data items, are all $O(1)$. In this case, the hardness will have to arise from the structure of the network.
- It is interesting to study the approximability of the problem of maximizing the number of happy connections.

## VI. Acknowledgement

This work was supported in part by the National Natural Science Foundation of China Grant 60553001, the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

## References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.
[2] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. *Information and Computation*, 96(1):77–94, 1992.
[3] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proc. of Allerton Conference on Communication, Control, and Computing*, 2003.
[4] R. Diestel. *Graph Theory*. Springer, 2005.
[5] R. Dougherty, C. Freiling, , and K. Zeger. Insufficiency of linear coding in network information flow. *IEEE Transactions on Information Theory*, 8:2745–2759, 2005.
[6] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6):1973–1982, 2003.
[7] M. Langberg and A. Sprintson. On the hardness of approximating the network coding capacity. In *Proc. of ISIT*, 2008.
[8] A. R. Lehman and E. Lehman. Complexity classification of network information flow problems. In *Proc. of SODA*, 2004.
[9] Z. Li, B. Li, and L. C. Lau. On achieving maximum multicast throughput in undirected networks. *IEEE Transactions on Information Theory*, 52(6):2467–2485, 2006.
[10] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.
[11] A. Wigderson. P, NP and mathematics - a computational complexity perspective. In *Proc. of ICM 06*, volume 1, pages 665–712, 2007.
[12] R. W. Yeung. *Information Theory and Network Coding*. Springer, 2008.
[13] R. W. Yeung, S.-Y. Li, and N. Cai. *Network Coding Theory*. Now Publishers, 2006.