Decoding Network Codes using the Sum-Product Algorithm

Anindya Gupta and B. Sundar Rajan
Dept. of ECE, IISc, Bangalore 560012, India, Email:{anindya.g, bsrajan}@ece.iisc.ernet.in

***Abstract*—While feasibility and obtaining a solution of a given network coding problem are well studied, the decoding procedure and complexity have not garnered much attention. We consider the decoding problem in a network wherein the sources generate multiple messages and the sink nodes demand some or all of the source messages. We consider both linear and non-linear network codes over a finite field and propose to use the sum-product (SP) algorithm over Boolean semiring for decoding at the sink nodes in order to reduce the computational complexity. We use *traceback* to further lower the computational cost incurred by SP decoding. We also define and identify a sufficient condition for *fast decodability* of a network code at a sink that demands all the source messages.**

I. Introduction

In contemporary communication networks, the nodes perform only routing, *i.e.*, they copy the data on incoming links to the outgoing links. In order to transmit messages generated simultaneously from multiple sources to multiple sinks the network may need to be used multiple times. This limits the throughput of the network and increases the time delay too. It is known that if intermediate nodes in a network are permitted to perform coding operations, *i.e.*, encode data received on the incoming links and then transmit it on the outgoing links (each outgoing link can get differently encoded data), the throughput of the network increases. This is called network coding [1]. Thus, network coding subsumes routing.

For example, consider the butterfly network [1] of Fig. 1 wherein each link can carry one bit per link use, source node S generates bits b_1 and b_2, and both sink nodes T_1 and T_2 demand both source bits. With routing only, two uses of link $V_3 - V_4$ are required while with network coding only one.

Above is an example of single-source multi-sink linear multicast network coding, wherein there is a single source (S), generating a finite number of messages, (x_1, x_2), and multiple sinks, each demanding all the source messages and the encoding operations at all nodes are linear. In general, there may be several source nodes, each generating different number of source messages, and several sink nodes, each demanding only a subset, and not all, of source messages. Decoding at sink nodes with such general demands is studied in this paper.

We represent a network by a finite directed acyclic graph $N = (V, E)$, where V is the set of vertices or nodes and $E \subseteq V \times V$ is the set of directed links or edges between nodes. All links are assumed to be error-free. Let $[n] = \{1, 2, \ldots, n\}$. The network has J sources, S_j, $j \in [J]$, and K sinks, T_k, $k \in [K]$. The source S_j generates ω_j messages, $\forall j \in [J]$. Let $\omega = \sum_{j=1}^{J} \omega_j$ be the total number of source messages. The ω-tuple of source messages is denoted by $x_{[\omega]} = (x_1, x_2, \ldots, x_\omega)$, where $x_i \in F$, $\forall i \in [\omega]$ and F is a finite field. By $\mathbf{x} = (x_1, \ldots, x_\omega)^T$ we denote the column vector of the source messages. The demand of the k^{th} sink node is denoted by $D_k \subseteq [\omega]$. Given a set $I = \{i_1 \ldots, i_l\} \subseteq [\omega]$, let $x_I = (x_{i_1}, \ldots, x_{i_l})$, *i.e.*, $x_{[\omega]}$ *restricted* to I. Let $\{x_I\} = \{x_I \colon x_I \in F^I\}$, *i.e.*, the set of all I-tuples over F. For a multi-variable binary-valued function $f(x_1, \ldots, x_\omega)$, the subset of F^ω whose elements are mapped to 1 by $f(x_1, \ldots, x_\omega)$ is called its support and is denoted by $\mathtt{supt}(f(x_{[\omega]}))$ and $\mathtt{supt}_I(f(x_{[\omega]}))$ denotes the $|I|$-tuples in the support restricted to I. A source message is denoted by edges without any originating node and terminating at a source node. Data on a link $e \in E$ is denoted by y_e.

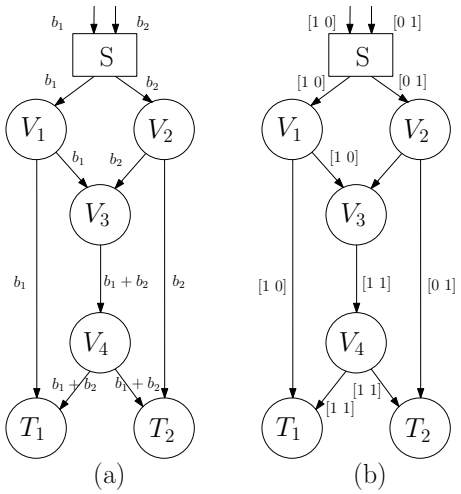

Fig. 1. Butterfly Network (a) A network code and (b) Global encoding vectors

A network code (NC) is a set of coding operations to be performed at each node such that the requisite source messages can be faithfully reproduced at the sink nodes. It can be specified using either local or global description [1]. The former specifies the data on a particular outgoing edge as a function of data on the incoming edges while the latter specifies the data on a particular outgoing edge as a function of source messages. Throughout the paper we use global description for our purposes.

Definition 1: Global Description of an NC [1]: An ω-dimensional NC on an acyclic network over a field F consist of $|E|$ global encoding maps $\tilde{f}_e : F^\omega \to F, \forall e \in E$ (i.e., $\tilde{f}_e(\mathbf{x}) = y_e$).

Let $e_i, i = 1, \ldots, \omega$, be the incoming edges at the source, then $y_{e_i} = x_i$.

When the intermediate nodes perform only linear encoding operations then such an NC is said to be a linear network code (LNC).

Definition 2: Global Description of an LNC [1]: An ω-dimensional LNC on an acyclic network over a field F consist of $|E|$ $1 \times \omega$ global encoding vectors $\mathbf{f}_e$, $\forall e \in E$ such that $\mathbf{f}_e \cdot \mathbf{x} = y_e$.

The global encoding vectors for the incoming edges at the source are standard basis vectors for the vector space F^ω. The global encoding vectors of the LNC for butterfly network is given in Fig. 1(b).

Hereafter we assume that the network is feasible, *i.e.,* demands of all sink nodes can be met using network coding and the global description of a network code (linear or non-linear) is given. If a sink node demands ω' ($\leq \omega$) source messages, it will have at least ω' incoming edges. The decoding problem is to reproduce the desired source messages from the coded data received at the incoming edges. Thus, decoding amounts to solving for a specified set of ω' unknowns using a set of at least ω' simultaneous equations in ω unknowns. Hence, the global description of the NC is more useful for decoding.

While decoding of non-linear NC has not been studied, the common technique used for decoding a LNC is to perform Gaussian elimination [2], [3], which requires $\mathcal{O}(n^3)$ operations, followed by backward substitution, which requires $\mathcal{O}(n^2)$ operations (n is the number of variables) [4]. This is not recommendable when the number of equations and/or variables is very large. In such cases, iterative methods are used. Convergence and initial guess are some issues that arise while using iterative methods [5].

We propose to use the sum-product (SP) algorithm to perform iterative decoding at the sinks. A similar scheme for decoding multicast network codes using factor graph [6] was studied in [7]. The authors considered the case of LNC. The problems associated with the proposed decoding scheme in [7] are:

- In order to construct the factor graph, full knowledge of network topology is assumed at the sinks which is impractical if the network topology changes. For a particular sink node (say T), the factor graph constructed will have $\omega + |E|$ variable nodes and $|E| + |In(T)|$ factor nodes, where $In(T)$ is the set of incoming edges at node T.
- Complete knowledge of local encoding matrix [1] of each node is assumed at the sinks which again is impractical since local encoding matrix for different nodes will have different dimensions and hence variable number of overhead bits will be required to communicate to downstream nodes which will incur huge overhead.

We also point out that the motivating examples, *viz.,* Examples 1 and 4, given in [7] for which the proposed decoding method claims to exploit the network topology admits a simple routing solution and no network coding is required to achieve maximum throughput. Solving linear equations in boolean variables is also studied in [8].

The contributions and organization of the paper are as follows:

- In Section III-A we pose the problem of decoding of linear and non-linear NC as *marginalize a product function problem* (MPF) and construct factor graph using the global description of network codes. For a particular sink node, the constructed graph will have fewer vertices than in [7] and hence the number of messages and operations performed will also be fewer. Unlike in [7], our scheme requires only the knowledge of global encoding maps/vectors of incoming edges at a sink node and not the entire network structure and coding operation performed at each node.
- In Sections III-B, we utilize *traceback* instead of running multiple-vertex version of algorithm, thus, further reducing the number of operations. Some examples illustrating the proposed techniques are given in Section III-D.
- We discuss utility and computational complexity of the proposed technique in Section III-C. For sink nodes which demand all the source messages, the notion of *fast decodable network codes* is defined and a sufficient condition for the same is identified.

We present a brief overview of the SP algorithm in Section II and conclude the paper with a discussion on scope for further work in Section IV.

II. The Sum-Product Algorithm and Factor Graphs

In this section, we review the computational problem called the MPF problem and specify how SP algorithm can be used to efficiently solve such problems. An equivalent method to efficiently solve MPF problems is given in [9] and is called the *generalized distributive law* (GDL) or the *junction tree algorithm*. The simplest example of SP algorithm offering computational advantage is the distributive law on real numbers, $a \cdot (b + c) = a \cdot b + a \cdot c$; the left hand side of the equation requires fewer operation than the right hand side. Generalization of addition and multiplication is what is exploited by the SP (or the junction tree) algorithm in different MPF problems. The mathematical structure in which these operations are defined is known as commutative semirings.

Definition 3: A commutative semiring [9] is a set R, together with two binary operations "$+$" (*addition*) and "$\cdot$" (*multiplication*), which satisfy the following axioms:

1) The operation "$+$" satisfies closure, associative, and commutative properties; and there exists an element "0" (*additive identity*) such that $r + 0 = r, \forall r \in R$.
2) The operation "$\cdot$" satisfies closure, associative, and commutative properties; and there exists an element "1" (*multiplicative identity*) such that $r \cdot 1 = r$, $\forall r \in R$.
3) The operation "$\cdot$" *distributes* over "$+$", *i.e.,* $r_1 \cdot r_2 + r_1 \cdot r_3 = r_1 \cdot (r_2 + r_3)$, $\forall r_1, r_2, r_3 \in R$

For different problems, we use different semirings with different notion of "$+$ and $\cdot$". Some examples are listed below.

1) Application of the SP algorithm to Fourier transform yields the FFT algorithm; the semiring is the set of complex numbers with the usual addition and multiplication [6], [9].
2) ML decoding of binary linear codes is also an MPF problem and application of SP algorithm yields the Gallager-Tanner-Wiberg decoding algorithm over a Tanner graph; the semiring is the set of positive real numbers with "$\min$" as sum and "$+$" as product, called the min-sum semiring [6], [9]. The BCJR algorithm for decoding turbo codes and LDPC deocoding algorithm are some other applications of SP algorithm.
3) Application to the ML sequence estimation, for instance in decoding convolutional codes, yields the Viterbi algorithm [9]; the semiring is again the min-sum semiring.
4) Recently, the GDL has been shown to reduce the ML decoding complexity of space-time block codes in [10]; the semiring applicable is the min-sum semiring of complex number. The authors introduced *traceback* for GDL and used it to further lower the number of operations.

Thus, both these algorithms subsume as special cases many well known algorithms.

A. MPF Problems in Boolean Semiring

A Boolean semiring is the set $\{0,1\}$ together with the usual Boolean operations $\vee$ (OR) and $\wedge$ (AND). We denote it by $R = (\{0,1\}, \vee, \wedge)$. The elements 0 and 1 are the *additive* and *multiplicative identities* respectively. The MPF problem defined for this semiring is described below. Let $x_i, i \in [n]$ be n variables taking values in finite alphabets $A_i, i \in [n]$. For $I = \{i_1, \ldots, i_k\} \subseteq [n]$, let $x_I = (x_{i_1}, \ldots, x_{i_k})$ $A_I = A_{i_1} \times \ldots \times A_{i_k}$. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_M\}$, $S_j \subseteq [n]$, such that for each $j \in [M]$, there is a function $\alpha_j : A_{S_j} \to R$. The functions α_js are called the *local kernels*, the set of variables in x_{S_j} is called the *local domain* associates with α_j and A_{S_j} is the associated *configuration space*. The *global kernel*, $\beta : A_{[n]} \to R$ and its i^{th} *marginalization*, $\beta_i : A_{S_i} \to R$, are defined below.

$$\beta(x_1, x_2, \ldots, x_n) = \bigwedge_{j=1}^{M} \alpha_j(x_{S_j})$$

$$\beta_i(x_i) = \bigvee_{\{x_{[n]\setminus i}\}} \beta(x_1, x_2, \ldots, x_n) \tag{1}$$

B. The SP Algorithm

Brute force computation of marginalizations (1) require $\mathcal{O}(A_{[n]})$ computations; the SP algorithm is an efficient way of computing these. It involves iteratively passing *messages* along the edges of the *factor graph*, $\mathcal{G} = (\mathcal{V} \cup \mathcal{F}, \mathcal{E})$, associated with the given MPF problem. The factor graph is a bipartite graph. Vertices in $\mathcal{V}$ are called variable nodes; one for each variable x_i, $\forall i \in [n]$ ($|\mathcal{V}| = n$). The vertices in $\mathcal{F}$ are called the factor nodes; one for each local kernel α_j, $\forall j \in [M]$ ($|\mathcal{F}| = M$). A variable node x_i is connected to a factor node α_j iff x_i is an argument of α_j. For convenience we assume that for a variable node x the local domain and local kernel are x and 1 respectively.

Let $N(x_i)$ denote the set of factor nodes adjacent to the variable node x_i, *i.e.*, set of local kernels with x as an argument and $N(\alpha_j)(= x_{S_j})$ denote the set of variable nodes adjacent to the factor node α_j, *i.e.*, the local domain of α_j. The directed message passed from a variable node x_i to an adjacent factor node α_j and vice versa are as follows:

$$\mu_{x_i \to \alpha_j}(x_i) = \bigwedge_{\alpha' \in N(x_i) \setminus \alpha_j} \mu_{\alpha' \to x_i}(x_i) \tag{2}$$

$$\mu_{\alpha_j \to x_i} = \bigvee_{\{x_{S_j \setminus i}\}} \alpha_j(x_{S_j}) \bigwedge_{x' \in \{x_{S_j \setminus i}\}} \mu_{x' \to \alpha_j}(x') \tag{3}$$

Depending on the requirement, we may need to evaluate marginal(s) at only one, a few or all variable nodes. The algorithm starts at the leaf nodes (nodes with degree one) with the leaves passing messages to the adjacent nodes. Once a vertex has received messages from all but one of its neighbor, it computes its own message and passes it to the neighbor from which it has not yet received the message. This message passing terminates when all the variable nodes at which marginals are required to be evaluated have received from all its neighbors. A node after receiving messages from all of its neighbors, computes its *state* as the product of messages received from all the adjacent nodes. For a variable node, x_i, it is denoted by $\sigma_i(x_i)$ and is given as follows:

$$\sigma_i(x_i) = \bigwedge_{\alpha' \in N(x_i)} \mu_{\alpha' \to x_i}(x_i), \tag{4}$$

Similarly, the state for a factor node is computed as follows:

$$\sigma_{\alpha_j}(x_{S_j}) = \alpha_j(x_{S_j}) \bigwedge_{x' \in \{x_{S_j}\}} \mu_{x' \to \alpha_j}(x'), \tag{5}$$

As stated in [6], [9], after sufficient number of messages have been passed, the state of a variable node x_i will be equal to $\beta_i(x_i)$.

To obtain the correct value of the required marginal(s), it is essential that the factor graph be free of cycles. If there are cycles these may not be the correct values. The cycles can be eliminated by *stretching* variable nodes or *clustering* variable or factor nodes (refer to [6, Sec. VI] for a detailed description). These methods are exemplified in Section III-D.

Both these graph transformations lead to enlargement of the local domain(s), and hence the configuration space of the node(s). In the new graph, the directed message passed from a vertex v to w is

$$\mu_{v \to w}(x_{S_v \cap S_w}) = \bigvee_{\{x_{S_v \setminus S_w}\}} \alpha_v(x_{S_v}) \bigwedge_{v' \in N(v) \setminus w} \mu_{v' \to v}(x_{S_{v'} \cap S_v}), \tag{6}$$

where $N(v)$ is the set of neighboring vertices of v and its state $\sigma_v(x_{S_v})$ is

$$\sigma_v(x_{S_v}) = \alpha_v(x_{S_v}) \bigwedge_{v' \in N(v)} \mu_{v' \to v}(x_{S_{v'} \cap S_v}) \tag{7}$$

These are the general forms of messages and states; (2)-(5) can be obtained from these.

Let v^* be the node with the largest configuration space A_{v^*} (choose any one if there are multiple such nodes). Then the number of operations required for computing messages and states in the SP algorithm will be $\mathcal{O}(A_{v^*})$. Thus, at the cost of possibly increased computational cost, the SP algorithm on the transformed graph yields the exact value of the marginals. In the sequel, we assume that the factor graph is acyclic.

III. Decoding Network Codes using the Sum-Product Algorithm

In this section, we show that decoding a NC is an MPF problem over a Boolean semiring. We provide a method to construct factor graph for decoding at a sink node using the SP algorithm.

Though the factor graph approach and the junction tree approach are equivalent formulations to solve MPF problems, we prefer the former because of the amount of preprocessing required to obtain junction tree as argued below:

1) The construction of a junction tree for an MPF problem requires: *(a)* construction of a *moral graph*, *(b)* its *minimum complexity triangulation* if it is not already triangulated, *(c)* construction of the *clique graph* of the triangulated moral graph, and *(d)* finding a spanning tree which leads to minimum computational cost. To the nodes of this clique tree the local kernels and variables of the MPF problem are attached [9] to obtain the junction tree (a kernel or a variable is attached to a node of clique tree iff its local domain is a subset of the local domain of the said clique tree node). Thus, the GDL always gives the exact solution of the MPF problems.
2) A factor graphs is easily described by the local kernels associated with the MPF problem; it is a bipartite graph involving set of variable and a set of local kernels of the MPF kernels as the two vertex sets. If it is acyclic, then the SP algorithm gives the exact solution, if not, it gives an approximate solution. The SP algorithm is known to perform well even if the factor graph has cycles, for example, in decoding of LDPC and turbo codes. Factor graphs with cycles can be transformed into acyclic ones to obtain exact solutions [6, Sec. VI].

A. NC Decoding as an MPF Problem

Given an acyclic network $N = (V, E)$, the demands at each sink, D_k, $k \in [K]$ and a set of global encoding maps, $\{\tilde{f}_e : e \in E\}$, that satisfy all the sink demands, the objectives at a sink, say k^{th}, is to find the instance of x_{D_k} that was generated by the source(s) using the data it receives on its incoming edges, *i.e.*,

$$x^*_{D_k} = \underset{D_k}{\texttt{supt}} \bigvee_{x_{[\omega]} \in F^\omega} \underbrace{\bigwedge_{e \in In(T_k)} \delta\left(\tilde{f}_e(x_{[\omega]})\,,\, y_e\right)}_{\beta^{(k)}(x_{[\omega]})} \tag{8}$$

Here $\beta^{(k)}$ is the global kernel of the MPF problem at the k^{th} sink and δ is a function that indicates whether its two input arguments are equal or not,*i.e.* ,

$$\delta(a, b) = \begin{cases} 0, & \text{if } a \neq b \\ 1, & \text{if } a = b \end{cases}$$

For LNC, (8) becomes

$$x^*_{D_k} = \underset{D_k}{\texttt{supt}} \bigvee_{\mathbf{x} \in F^\omega} \bigwedge_{e \in In(T_k)} \delta\left(\mathbf{f}_e \cdot \mathbf{x}\,,\, y_e\right)$$

Thus, decoding a NC has the form of a special class of MPF problems over Boolean semiring wherein we are interested only in some coordinates (specified by D_k) of the ω-tuples in the support set and not the value of the global kernel.

Since the solution $x^*_{D_k}$ is unique, individual coordinates $j \in D_k$ can be separately computed, *i.e.*,

$$\begin{aligned} x^*_j &= \underset{j}{\texttt{supt}} \bigvee_{x_j \in F} \beta^{(k)}_j(x_j) \\ \beta^{(k)}_j(x_j) &= \bigvee_{\{x_{[\omega]\setminus j}\}} \beta^{(k)}(x_{[\omega])}, \end{aligned} \tag{9}$$

where $\beta^{(k)}_j(x_j)$ is the j^{th} marginalization of the global kernel $\beta^{(k)}$.

The factor graph for decoding at sink $T_k, k \in [K]$ is constructed as follows:

1) Install ω *variable nodes*, one for each source message. These vertices are labeled by their corresponding source messages, x_i.
2) Install $|In(T_k)|$ *factor nodes* and label them $\tilde{f}_e, e \in In(T_k)$. The associated local domain of each such vertex is the set, $S \subseteq x_{[\omega]}$, of source messages that participate in that encoding map and the local kernel is $\delta(\tilde{f}_e(x_{[\omega]})\,,\, y_e)$. These vertices are labeled by their corresponding local kernels, $\tilde{f}_e$.
3) A variable node is connected to a factor node iff the source message corresponding to that variable node participates in the encoding map corresponding to the said factor node.

We use thicker lines for factor nodes to differentiate them from variable nodes. The factor graph so constructed will be a bipartite graph. General form of a factor graph and the same for the two sink nodes of the butterfly network are given in Fig. 2 (cf. [7, Fig. 3]).

Messages and states are computed using (6) and (7) respectively. As stated before, once a node (say v), has received message from all the adjacent nodes, its state, $\sigma_v(x_{S_v})$, can be computed. Let $x_{D_k} \cap x_{S_v} = x_B$. The value of the subset B of the requisite source messages at the k^{th} sink node is

$$x^*_B = \underset{B}{\texttt{supt}} \bigvee_{\{x_{S_v}\}} \alpha_v(x_{S_v}) \bigwedge_{v' \in N(v)} \mu_{v' \to v}(x_{S_{v'} \cap S_v}) \tag{10}$$

As specified in Section II, the SP algorithm yields the correct value of the source messages if the factor graph is a tree. If not then the cycles in the factor graph will have to be removed.

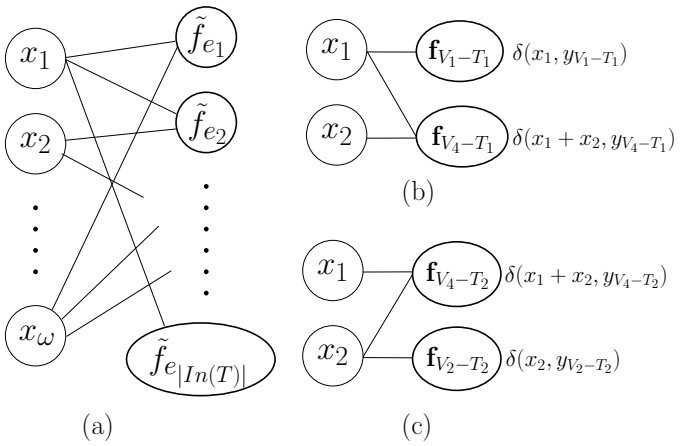

Fig. 2. (a) General form of a factor graph (b) Factor graphs for T_1 and (c) T_2 of the butterfly network. Local kernels are given adjacent to the function vertices.

B. Traceback

Since we are interested in the value of the source messages and not the value of the marginalizations of the global kernel, we can use *traceback* [10] to further lower the number of computations.

Assume that the single-vertex SP algorithm is run with vertex v as the root (all messages are directed towards v) and the value $x^*_{S_v} \subseteq x_{D_k}$ of some source messages in the demand set of the k^{th} sink has been ascertained. Now, partition the local domain of a neighboring node w, as $x_{S_w} = x_A \cup x_B$, where $A = S_w \backslash S_v$ and $B = S_w \cap S_v$. Since $x^*_{S_v}$ is known, the value x^*_B that causes $\sigma_w(x_A, x_B)$ to take value 1 is also known. The value of the source messages x^*_A can then be obtained using (7) as follows:

$$\begin{aligned} x^*_A &= \underset{A}{\mathtt{supt}} \bigvee_{\{x_A\}} \sigma_w(x_A, x^*_B) \\ &= \underset{A}{\mathtt{supt}} \bigvee_{\{x_A\}} \mu_{v\to w}(x^*_B)\ \lambda_w(x_A, x^*_B) \\ &= \underset{A}{\mathtt{supt}} \bigvee_{\{x_A\}} \lambda_w(x_A, x^*_B), \end{aligned} \tag{11}$$

where

$$\lambda_w(x_A, x_B) = \alpha(x_{S_w}) \bigwedge_{v' \in N(w)\backslash v} \mu_{v'\to w}(x_{S_{v'} \cap S_w})$$

is the partial state computed at w while passing the message $\mu_{w\to v}(x_B)$ to the root v. Thus, $\mu_{v\to w}(x^*_B)$ does not need to be computed or passed, leading to saving of operations.

Here we have exploited the fact that we require only $\mathtt{supt}_A \bigvee_{x_A} \sigma_w$ and not the value of σ_w which would have required passing of the message $\mu_{v\to w}(x^*_B)$ from v to w too. Hence, the traceback step reduces the computational complexity.

The traceback step can be used repeatedly until values of all the source messages in x_{D_k} are obtained. This can be done by using (11) on other neighbors of v and then neighbors of neighbors of v and so on. This can lead to considerable reduction in number of operations and is exemplified in Section III-D.

C. Computational Complexity

We suggest using SP algorithm for decoding a network code only when the code is either non-linear or it is linear but the number of messages is very large. For linear network codes with manageable value of ω, Gaussian elimination with backward substitution is advisable. If using SP algorithm for decoding network codes (when warranted) leads to computational complexity strictly better than brute-force decoding complexity, then the code is called *fast decodable network code*.

We now discuss the computational complexity of SP algorithm based decoding of network codes. As in [6], by complexity we mean the number of semiring operations required to obtain the desired source message. For convenience, we assume that all the source messages take value from a q-ary alphabet; results for variable alphabet size can be obtained similarly.

As stated above, in order to recover the requisite source messages at a sink we need only run single-vertex SP algorithm followed by traceback steps. For a given sink node, if the factor graph constructed using the method given in Section III-A is cycle-free and the network code is such that the local domains of all factor nodes have cardinality at most $l\,(<\omega)$, then the number of operations required for decoding using the SP algorithm is $\mathcal{O}(q^l)$. If the sink demands all the source messages, then the brute-force decoding would require $\mathcal{O}(q^\omega)(>\ \mathcal{O}(q^l))$ operations. Thus, an acyclic factor graph with at most $l\,(<\ \omega)$ variables per equation is a sufficient condition for fast decodability of the network code at a sink which demands all the source messages.

If the graph is not cycle-free then we remove the cycles using the methods specified above and let $m \leqslant \omega$ be the size of maximum cardinality local domain in the transformed cycle-free factor graph. The number of computations required now will be $\mathcal{O}(q^m) \leqslant \mathcal{O}(q^\omega)$ and the code is fast decodable iff $m < \omega$.

D. Illustrations

We now present some examples illustrating use of the SP algorithm to decode NC.

Example 1: Consider the butterfly network of Fig.1. Here $q = \omega = 2$. The factor graphs for two sink nodes are given in are given in Fig. 2(b) and (c). The messages passed and state computations for decoding at T_1 are as follows:

$$\begin{aligned} \mu_{x_2\to \mathbf{f}_{V_4-T_1}}(x_2) &= 1 \\ \mu_{\mathbf{f}_{V_1-T_1}\to x_1}(x_1) &= \delta(x_1, y_{V_1-T_1}) \\ \mu_{\mathbf{f}_{V_4-T_1}\to x_1}(x_1) &= \bigvee_{x_2} \delta(x_1+x_2, y_{V_4-T_1}) \\ \mu_{x_1\to \mathbf{f}_{V_4-T_1}}(x_1) &= \mu_{\mathbf{f}_{V_1-T_1},x_1}(x_1) \\ \sigma_{x_1}(x_1) &= \mu_{\mathbf{f}_{V_1-T_1},x_1}(x_1)\ \mu_{\mathbf{f}_{V_4-T_1}\to x_1}(x_1) \end{aligned}$$

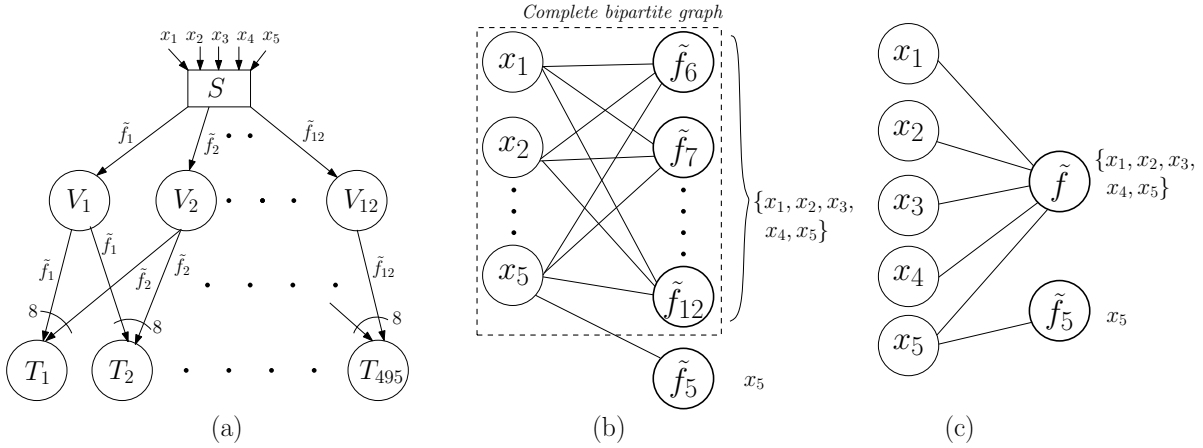

Fig. 3. (a) Combination network (b) Factor graph with cycles for T_{495} and (c) one obtained by clustering equation nodes $\tilde{f}_6 - \tilde{f}_{12}$ ($\tilde{f} = \bigwedge_{i=6}^{12} \tilde{f}_i$)

$$\mu_{\mathbf{f}_{V_4-T_1}\to x_2}(x_2) = \bigvee_{x_1} \delta(x_1 + x_2, y_{V_4-T_1})\ \mu_{\mathbf{f}_{V_1-T_1}\to x_1}(x_1)$$
$$\sigma_{x_2}(x_2) = \mu_{\mathbf{f}_{V_4-T_1},x_2}(x_2)$$

It is easy to verify that $\sigma_{x_i}(x_i) = \beta_i(x_i), i = 1, 2$. Hence, $x_i^* = \texttt{supt}_{x_i} \bigvee_{x_i} \sigma_{x_i}(x_i), i = 1, 2$. Similar computations apply for T_2 also. This network code is not fast decodable since the number of computations required is $\mathcal{O}(q^\omega)$. □

Now we present a multicast network which admits no linear solution over $\mathbb{F}_2$, but a non-linear solution exists.

Example 2: Consider the multicast combination network given in Fig.3(a). All messages take value from F_2. Each sink is connected to a distinct size 8 subset of intermediate node. Multicast, in F_2, is feasible over such a network iff a $(12, 32, 5)$ binary error correcting code exists [11]. The Nadler's code is one such systematic code with the requisite parameters. Note that since there exist no $(12, 32, 5)$ binary linear code, the above multicast network, admits no linear solution over F_2.

Apart from the systematic part of the Nadler's code, the 7 redundant bits are encoded using non-linear functions [12];these are:

$$\begin{aligned}
\tilde{f}_i &= x_i, \quad i \in [5] \\
\tilde{f}_6 &= x_1 + x_2 + x_3 + (x_1 + x_5)(x_3 + x_4) \\
\tilde{f}_7 &= x_1 + x_2 + x_4 + (x_1 + x_3)(x_4 + x_5) \\
\tilde{f}_8 &= x_1 + x_2 + x_5 + (x_1 + x_4)(x_3 + x_5) \\
\tilde{f}_9 &= x_2 + x_3 + x_4 + x_1x_4 + x_4x_5 + x_5x_1 \\
\tilde{f}_{10} &= x_2 + x_3 + x_5 + x_1x_3 + x_3x_5 + x_5x_1 \\
\tilde{f}_{11} &= x_2 + x_4 + x_5 + x_1x_3 + x_3x_5 + x_5x_1 \\
\tilde{f}_{12} &= x_1 + x_2 + x_3 + x_4 + x_5 + x_3x_4 + x_4x_5 + x_5x_3
\end{aligned}$$

These functions are the global encoding maps of the 12 source to intermediate node links. The intermediate nodes simply route the data on incoming edges to the connected sink nodes, hence, the global encoding maps of the incoming and outgoing edges of an intermediate node is same.

Consider decoding at the 495^{th} sink whose global encoding maps of incoming edges are $\tilde{f}_i,\ i = 5, \ldots, 12$. The factor graph constructed using the method stipulated above will have cycles (Fig. 3(c)). To eliminate the cycles, we cluster $\tilde{f}_i,\ i = 6, \ldots, 12$ function vertices into a single one with the local kernel as the product of 7 original ones (Fig. 3(c)). The computational complexity of SP decoding is same as that of brute-force decoding ($\mathcal{O}(q^5)$). This is true for all the sink nodes and hence this code is not fast decodable for any of the sinks. □

In the next example we present a network with general demands at sinks and employ the SP algorithm for decoding a vector non-linear network code for it. We also demonstrate usefulness of traceback for saving computations of some messages in the factor graph.

Example 3: Consider the network given in Fig. 4. The sinks (nodes $37 - 46$) have general demands which are specified by variables below them. In [13], the authors showed that this network admits no linear solution over any field and gave a vector non-linear solution. The source message $x_i, i \in [5]$ are 2-bit binary words ($q = 4, \omega = 5$), $+$ denotes addition in ring $\mathbf{Z}_4$, $\oplus$ denotes the bitwise XOR and the function $t(\cdot)$ reverses the order of the 2-bit input.

The factor graphs for nodes 37, 40 and 43, denoted by $\mathcal{G}_{37}, \mathcal{G}_{40}$ and $\mathcal{G}_{43}$ respectively, are given in Fig. 5; the same for other nodes have similar structure and can be constructed using method given in Section III-A. Note that $\mathcal{G}_{40}$ has a cycle of length 4 and $\mathcal{G}_{43}$ has two cycles of length 6 each. The cycles are removed as follows:

- The cycle in $\mathcal{G}_{40}$ is removed by clustering the variable vertices x_1 and x_2; the local domain and kernel of the new variable vertex are (x_1, x_2) (union of the local domains of the clustered vertices) and 1 (product of the local kernels of the clustered vertices) (Fig. 5).
- The cycles C_1 and C_2 in $\mathcal{G}_{43}$ are removed by deleting the dotted edges and *stretching* variable vertex x_3 around the respective cycles; this involves adding the stretched variable to all the local domain in the cycle and leaving

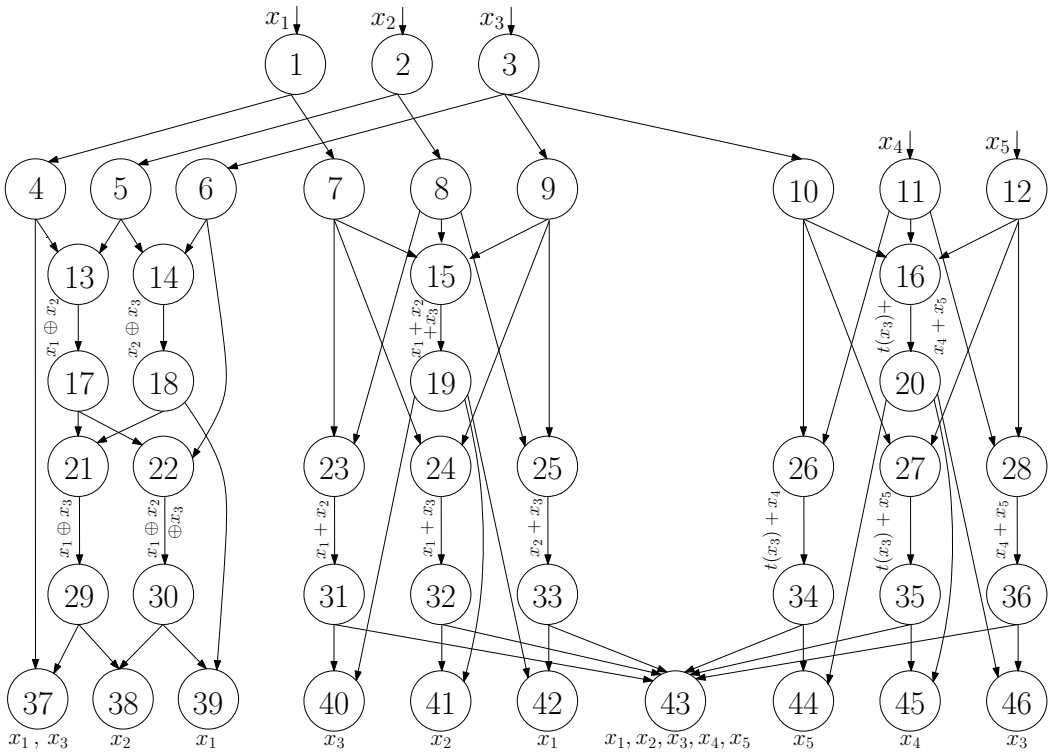

Fig. 4. The network $\mathcal{N}_3$ of [13].

the local kernels unchanged (Fig. 5).

We infer from $\mathcal{G}_{43}$ that the number of computations required to reproduce all the source messages at V_{43} is only $\mathcal{O}(q^3)$ instead of $\mathcal{O}(q^5)$ (as brute-force decoding would have required) and hence this code is fast decodable for V_{43}. The decoding process at V_{43}, using single-vertex SP algorithm with node "f" in $\mathcal{G}_{43}$ as root followed by traceback to compute x_1, x_2, x_4, x_5, is demonstrated below:

$$\mu_{k\to j} = \mu_{j\to i}(x_3, x_5) = \delta(t(x_3) + x_5, y_{35-43})$$
$$\mu_{a\to b} = \mu_{b\to c}(x_2, x_3) = \delta(x_2 + x_3, y_{33-43})$$
$$\mu_{i\to h} = \mu_{h\to g}(x_3, x_4) = \bigvee_{x_5} \delta(x_4 + x_5, y_{36-43})\mu_{j\to i}(x_3, x_5)$$
$$\mu_{c\to d} = \mu_{d\to e}(x_1, x_3) = \bigvee_{x_2} \delta(x_1 + x_2, y_{31-43})\mu_{b\to c}(x_1, x_2)$$
$$\mu_{e\to f}(x_3) = \bigvee_{x_1} \delta(x_1 + x_3, y_{32-43})\mu_{d\to e}(x_1, x_3)$$
$$\mu_{g\to f}(x_3) = \bigvee_{x_4} \delta(t(x_3) + x_4, y_{34-43})\mu_{h\to g}(x_3, x_4)$$

At "f", decoding of x_3 is performed as follows:

$$x_3^* = \operatorname*{supt}_{x_3} \bigvee_{x_3} \sigma_f(x_3) = \operatorname*{supt}_{x_3} \sigma_f(x_3)$$
$$= \operatorname*{supt}_{x_3} \mu_{e\to f}(x_3)\, \mu_{g\to f}(x_3)$$

We can now use traceback (11) to compute x_1 and x_2 without having to compute $\mu_{f\to e}, \mu_{e\to d}$ or $\mu_{d\to c}$ as follows:

$$x_1^* = \operatorname*{supt}_{x_1} \bigvee_{x_1} \sigma_e(x_1, x_3^*) = \operatorname*{supt}_{x_1} \lambda_e(x_1, x_3^*)$$
$$= \operatorname*{supt}_{x_1} \delta(x_1 + x_3^*, y_{32-43})\mu_{d\to e}(x_1, x_3^*)$$
$$x_2^* = \operatorname*{supt}_{x_2} \bigvee_{x_2} \sigma_c(x_1^*, x_2, x_3^*) = \operatorname*{supt}_{x_2} \lambda_c(x_1^*, x_2, x_3^*)$$
$$= \operatorname*{supt}_{x_2} \delta(x_1^* + x_2, y_{31-43})\mu_{b\to c}(x_2, x_3^*)$$

Note that $\mu_{d\to e}$ and $\mu_{b\to c}$ were already computed. Without traceback, x_1 and x_2 are decoded at variable nodes "d" and "b" respectively as follows:

$$\mu_{f\to e}(x_3) = \mu_{g\to f}(x_3)$$
$$\mu_{e\to d}(x_1, x_3) = \mu_{f\to e}(x_3)\delta(x_1 + x_3, y_{32-43})$$
$$x_1^* = \operatorname*{supt}_{x_1} \bigvee_{x_3} \mu_{e\to d}(x_1, x_3)\mu_{c\to d}(x_1, x_3)$$
$$\mu_{d\to c}(x_1, x_3) = \mu_{e\to d}(x_1, x_3)$$
$$\mu_{c\to b}(x_2, x_3) = \bigvee_{x_1} \mu_{d\to c}(x_1, x_3)\delta(x_1 + x_2, y_{31-43})$$
$$x_2^* = \operatorname*{supt}_{x_2} \bigvee_{x_3} \mu_{c\to b}(x_2, x_3)\mu_{a\to b}(x_2, x_3)$$

Similarly x_4 and x_5 can be obtained. The number of semiring operations required to compute all the messages passed and states computed is given in Table I.

The number of computations required with traceback are 2 (C1+C2+...+C5)+C6+C7+C8 which is $2(q^3+q^2)+5q = 180$

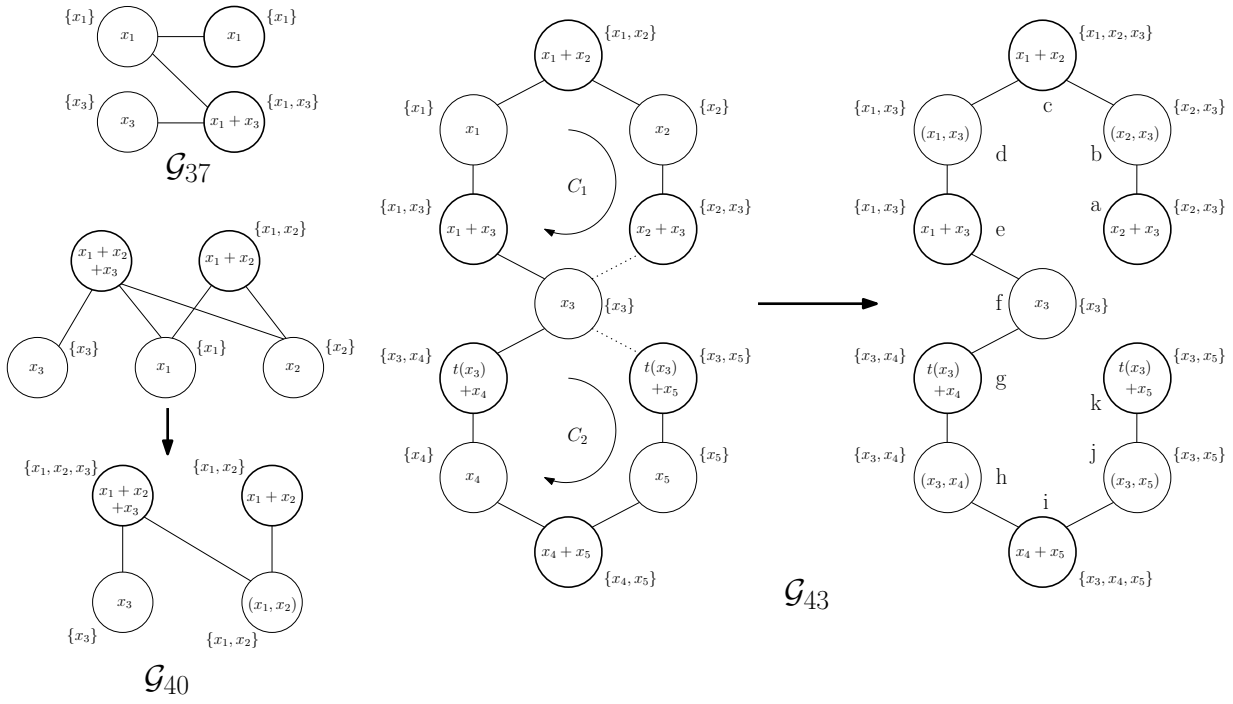

Fig. 5. The factor graphs for sinks with labels 37, 40 and 43 in network of Fig. 4.

	Messages/States	**No. of $\bigwedge$**	**No. of $\bigvee$**
C1	$\mu_{k\to j}, \mu_{j\to i}$	0	0
C2	$\mu_{a\to b}, \mu_{b\to c}$	0	0
C3	$\mu_{i\to h}, \mu_{c\to d}$	q^3	$q^2(q-1)$
C4	$\mu_{h\to g}, \mu_{d\to e}$	0	0
C5	$\mu_{e\to f}, \mu_{g\to f}$	q^2	$q(q-1)$
C6	x_3^*	q	0
C7	x_1^*, x_4^*	q	0
C8	x_2^*, x_5^*	q	0
C9	$\mu_{f\to e}, \mu_{f\to g}$	0	0
C10	$\mu_{e\to d}, \mu_{g\to h}$	q^2	0
C11	x_1^*	q^2	$q(q-1)$
C12	$\mu_{d\to c}, \mu_{h\to i}$	0	0
C13	$\mu_{c\to b}, \mu_{i\to j}$	0	0
C14	x_2^*	q^2	$q(q-1)$

TABLE I

products ($\wedge$) and $2(q^2(q-1)+q(q-1)) = 120$ sums ($\vee$).Without traceback, the number of operations required are 2(C1...+C5)+C6+2(C9+C10)+C11+2(C12+C13)+C14 which is $2q^3+6q^2 = 224$ products and $2q^2(q-1)+4q(q-1) = 144$ sums. Thus, running single-vertex SP algorithm followed by traceback step affords computational advantage over multiple-vertex version.

□

IV. Discussion

In this paper, we proposed a SP algorithm based decoder for decoding NC. Subsequently, a method for constructing the factor graph for a given sink node using the global encoding maps (or vectors in case of LNC) of the incoming edges and demands of the sink was provided. The graph so constructed had fewer nodes and led to fewer message being passed lowering the number of operations as compared to the scheme of [7]. Next we discussed how cycles in factor graph affect the solution of the MPF problem and illustrated with examples how to circumvent them. We introduced and discussed the advantages of traceback over multiple-vertex SP algorithm. Next, for the sinks demanding all the source messages, we introduced the concept of fast decodable network codes and provided a sufficient condition for a network code to be fast decodable.

References

[1] R. W. Yeung, *Information Theory and Network Coding*. Springer, 2008.

[2] D. S. Lun, M. Médard, R. Köetter and M. Effros, "On Coding for Reliable Communication over Packet Networks," Physical Communication 1, 1 (March 2008), 3-20.

[3] D. S. Lun, M. Médard, R. Köetter and M. Effros, "Further Results on Coding for Reliable Communication over Packet Networks," in Proc. ISIT 2005, pp. 1848-1852, 4-9 Sept. 2005.

[4] P. G. Ciarlet, *Introduction to Numerical Linear Algebra and Optimisation*, Cambridge University Press, 1989.

[5] G. Strang, *Introduction to Linear Algebra*, 3^{rd} Edn., Wellesley-Cambridge Press, 2003.

[6] F. R. Kschischang, B. J. Frey and H.-A. Loeliger, "The Generalized Distributive Law," IEEE Trans. Inf. Theory, vol. 46, no. 2, pp. 325-343, Mar. 2000.

[7] D. Salmond, A. Grant, T. Chan and I. Grivell, "Decoding Network Code by Message Passing," in Proc. IEEE ISIT 2009, pp. 423-427, June 28-July 3, 2009.

[8] M. Mézard and A. Montanari, *Information, Physics, and Computation*, Oxford University Press, 2009.

[9] S. M. Aji and R. J. McEliece," The Generalized Distributive Law," IEEE Trans. Inf. Theory, vol. 46, no. 2, pp. 325-343, Mar. 2000.

[10] L. P. Natarajan and B. S. Rajan, “Generalized Distributive Law for ML Decoding of Space-Time Block Codes,” IEEE Trans. Inf. Theory, vol. 59, no. 5, pp.2914-2935, May 2013.
[11] S. Riis and R. Ahlswede, “Problem in network coding and error-correcting codes,” in Proc. NetCod, Riva del Garda, Italy, 2005.
[12] C. Munuera and M. Barbier, “Wet paper codes and the dual distance in steganography,” Advances in Mathematics of Communications. vol. 6, no. 3, pp. 237 - 285, Aug. 2012.
[13] R. Dougherty, C. Freiling and K. Zeger, “Insufficiency of Linear Coding in Network Information Flow,” IEEE Trans. Inf. Theory, vol. 51, no. 8, pp.2745-2759, Aug. 2005.