Elimination of Cyclic Stopping Sets for Enhanced Decoding of LDPC Codes

Anxiao (Andrew) Jiang
Computer Science and Engineering Department
Texas A&M University
College Station, TX 77843-3112, USA
ajiang@cse.tamu.edu

***Abstract*—The Stopping-Set Elimination Problem is studied for LDPC codes: how to remove the fewest number of erasures from a stopping set such that the remaining erasures can be decoded by belief propagation in k iterations (including $k = \infty$). The problem is known to be NP-hard. Here efficient exact algorithms and approximation algorithms are presented for stopping sets whose induced graphs in Tanner graphs contain cycles.**

I. Introduction

This work studies a basic theoretical problem for LDPC codes introduced in [3]: *when the erasures in an LDPC codeword cannot be corrected by the decoder, how to remove the fewest number of erasures so that the remaining erasures become decodable?* Here we consider the widely-used iterative belief-propagation (BP) decoder: in each iteration, it uses every parity-check equation involving exactly one erasure to correct that erasure; and it stops when every equation involves zero or at least two erasures. When decoding fails, the undecodable erasures we are left with are called a *stopping set*. A stopping set and its associated parity-check nodes in the Tanner graph can be represented by a bipartite graph $G = (V \cup C, E)$ as follows: V represents a set of variable nodes that are erasures, C represents a set of check nodes each of which is adjacent to at least two nodes in V in the Tanner graph, and E represents the set of edges between them in the Tanner graph. Here V is a *Stopping Set* and G is a *Stopping Graph*. (An example of a stopping graph is shown in Fig. 1.) Now let k be a positive integer. The problem we study, the *Stopping-Set Elimination Problem* called SSE_k, is defined as follows [3], which differs from the many existing nice works on stopping sets [2], [4].

Definition 1. *Given a Stopping Graph $G = (V \cup C, E)$, how to remove the minimum number of variable nodes from V such that the remaining variable nodes can be decoded by BP decoding within k iterations?*

Here the parameter k controls the *required time* for decoding. If we consider only decodability but not decoding time, k can be seen as ∞ and we call the problem the SSE_∞ Problem.

The Stopping Set Elimination Problem has a number of applications in data storage and communications. Consider n blocks of data B_1, B_2, $\cdots$, B_n, where each block $B_i = (b_{i,1}, b_{i,2}, \cdots, b_{i,m})$ has m bits. Let them be m parallel LDPC codewords: for $1 \leq j \leq m$, the n bits $(b_{1,j}, b_{2,j}, \cdots, b_{n,j})$ form an LDPC codeword. (The mn bits can be seen as an $n \times m$ array, where each row is a block, and each column is an LDPC codeword.) When some blocks are lost (i.e., erased), they can be recovered by running the BP decoding algorithm identically on the m binary LDPC codewords in parallel. LDPC codes are an attractive option for erasure correction due to their high rates, simple XOR-based decoding and good locality (due to the low degrees of variable and check nodes).

If too many blocks are erased and BP decoding fails, extra measures can be used to retrieve some erased blocks from other sources so that the remaining erasures become decodable. Two such examples are shown in [3]. The first example is for distributed data storage, where the n blocks are distributively stored on n servers at a company's data center. If too many blocks are lost, the company can retrieve some blocks from its backup system – which is often at a remote location – so that the remaining erased blocks can be decoded locally. (Most big IT companies have such backup systems.) The other example is for satellite-to-ground communication, where too many lost packets (i.e., erased blocks) to the user triggers the user's request for the satellite to retransmit some of those lost packets. Since retrieving erased blocks from remote sites (e.g., remote backup system or satellite) is much more expensive than decoding data locally, it is desirable to minimize the number of retrieved blocks. This is the Stopping Set Elimination Problem.

When a set of variable nodes $S \subseteq V$ is removed from the stopping graph G, if the remaining variable nodes become decodable (we shall also say "the remaining graph is decodable"), then we call S an *Elimination Set*. So the SSE_k Problem seeks a minimum-sized (i.e., optimal) elimination set. In [3], the SSE_k Problem is proved to be NP-hard, even when k is very large ($k = \infty$) or small ($k = 1$). When the degrees of the variable nodes and check nodes are upper bounded by d_v and d_c, respectively, an approximation algorithm of approximation ratio $d_v(d_c - 1)$ and time complexity $O(d_v^2 d_c^2 V)$ is presented for the SSE_1 Problem. When the stopping graph is a tree (called a *Stopping Tree*), two efficient algorithms of time complexity $O(V + C)$ are presented for the SSE_∞ and SSE_k Problems, respectively, which finds an optimal elimination set in the stopping tree.

The stopping graph can be trees when the number of

erasures is not too many. However, a more general case is that the stopping graph can be cyclic but its number of cycles is bounded. That also matches the most useful cases in practice, where the number of erasures exceeds but is not far away from the LDPC code's decoding threshold. So we focus on this more general case in this paper. Specifically, we focus on p-Cyclic Stopping Graphs defined as follows.

Definition 2. *For a Stopping Graph $G = (V \cup C, E)$, if at most p variable nodes can be removed to make the remaining graph acyclic, then G is called a p-Cyclic Stopping Graph.*

The main results of the paper are as follows. In Section II, we analyze properties of the SSE_∞ Problem and present a scheme that finds optimal elimination sets for stopping graphs. We present an efficient linear-complexity algorithm based on the scheme for stopping graphs containing one cycle. We also present an approximation algorithm for the SSE_∞ Problem for p-cyclic stopping graphs, which has approximation ratio $\frac{2p}{c+1} + 1$ and polynomial time complexity, where $c \geq 0$ is an integer parameter that can be selected freely. In Section III, we present an extended polynomial-time approximation algorithm with the same approximation ratio for the SSE_k Problem for p-cyclic stopping graphs. We further analyze the SSE_k Problem for dense stopping graphs, and show that for high-rate LDPC codes with high erasure rates, all algorithms can achieve good approximation ratios.

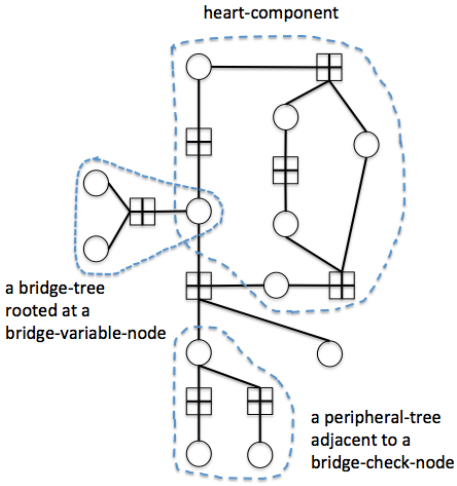

Fig. 1. Example of a stopping graph $G = (V \cup C, E)$.

II. SSE_∞ Problem for 1-Cyclic and p-Cyclic Stopping Graphs

In this section, we study the SSE_∞ Problem. We first analyze the properties of stopping graphs, and present an (not necessarily polynomial-time) algorithm that finds an optimal elimination set for any stopping graph. We then present two polynomial-time algorithms for stopping graphs with one cycle and for p-cyclic stopping graphs, respectively, which are variations of the first algorithm.

A. Properties of Stopping Graphs and SSE_∞ Problem

Let us first define several concepts. We call a graph *smooth* if every node in it is of degree at least two. We call a node in a graph a *smooth-node* if it is either in a cycle, or on a simple path that connects two cycles in the graph. We call a node v in a graph G a *tree-node* if v has degree one, or if removing v from G will break G into two or more disjoint subgraphs and at least one of those subgraphs is a tree that has only one node adjacent to v in G. Given a stopping graph $G = (V \cup C, E)$, if a node v in it is both a smooth-node and a tree-node, then we call v a *bridge-node*; furthermore, if $v \in V$ is a variable node (resp., if $v \in C$ is a check node), then we call v a *bridge-variable-node* (resp., a *bridge-check-node*). We let $\Lambda(G)$ denote the set of bridge-variable-nodes of G, and let $\Pi(G)$ denote the set of bridge-check-nodes of G.

Given a stopping graph $G = (V \cup C, E)$, let $\mathcal{S}_G$ denote the subgraph of G obtained by removing from G all the nodes that are tree-nodes but not bridge-nodes. We call $\mathcal{S}_G$ the *smooth-component* of G. Furthermore, let $\mathcal{C}_G$ denote the subgraph of $\mathcal{S}_G$ obtained by removing from $\mathcal{S}_G$ all the nodes in $\Pi(G)$. We call $\mathcal{C}_G$ the *heart-component* of G.

Lemma 3. *For any stopping graph $G = (V \cup C, E)$, $\mathcal{S}_G$ is a smooth graph. All the cycles in G are also in $\mathcal{S}_G$, and vice versa. The nodes in $\mathcal{S}_G$ are exactly those smooth-nodes in G. Furthermore, both $\mathcal{S}_G$ and $\mathcal{C}_G$ are stopping graphs (i.e., every check node in them is of degree at least two).*

Proof: For succinctness, we focus on the last property. (The other properties are easy to see.) When nodes are removed from G to get $\mathcal{S}_G$ or $\mathcal{C}_G$, for $\mathcal{S}_G$, every node has at least two neighbors since $\mathcal{S}_G$ is smooth; for $\mathcal{C}_G$, every check node in it has the same set of neighbors as it does in G, so its degree remains unchanged and is at least two. ■

Consider a stopping graph $G = (V \cup C, E)$ and a bridge-node v in it. By definition, removing v from G will break G into multiple subgraphs, at least one of which is a tree with exactly one node (say node u) adjacent to v in G. We call such a tree a *peripheral-tree* $\mathcal{T}_{pt,G}(u)$ in G, and call u its root. Let $\mathcal{A}_G(v)$ denote the set of roots of peripheral trees adjacent to v in G. Let $\mathcal{T}_{bt,G}(v)$ denote the subgraph of G containing these nodes: v, and all the nodes of those peripheral trees whose roots are in $\mathcal{A}_G(v)$ (namely, are adjacent to v in G). It can be seen that $\mathcal{T}_{bt,G}(v)$ is a tree, which we call a *bridge-tree*, and call v its root. It is not hard to see that G contains a smooth-component and a set of disjoint bridge-trees, where each bridge-tree overlaps the smooth-component at its root (a bridge node). See Fig. 1 for an illustration.

We will use the two algorithms in [3] that find optimal elimination sets for stopping trees – one algorithm for the SSE_∞ Problem and the other for the SSE_k Problem – in the algorithms of this paper. Due to space limitation, we do not include their details, and use them only as modules. For convenience, let us call the two algorithms the $Tree_\infty$ *Algorithm* and $Tree_k$ *Algorithm*, respectively. We note that the $Tree_\infty$ Algorithm has a property: *it can choose any variable*

node v in the stopping tree and ensure that the optimal elimination set it finds contains v. (v was seen as the tree's root in [3].) That property will be used in our future analysis.

We now present a recursive algorithm (called $OPT_\infty(G)$) that finds an optimal elimination set for any stopping graph for the SSE_∞ Problem. It takes a stopping graph $G = (V \cup C, E)$ as input, and outputs an elimination set $\mathcal{E} \subseteq V$ for G.

Algorithm 4. $OPT_\infty(G)$*:*

1) *If G is smooth, find an optimal elimination set $\mathcal{E} \subseteq V$ for G, and return $\mathcal{E}$.*[1] *Otherwise, go to Step 2.*
2) $\mathcal{E}_{heart} \leftarrow OPT_\infty(\mathcal{C}_G)$. *(Namely, use the heart-component $\mathcal{C}_G$ as input to this algorithm to obtain an elimination set $\mathcal{E}_{heart}$ for $\mathcal{C}_G$.)*
3) *For every bridge-node $v \in \Lambda(G) \cup \Pi(G)$, use the $Tree_\infty$ Algorithm to find an optimal elimination set $\mathcal{F}(v)$ for the bridge-tree $T_{bt,G}(v)$.*
4) *Return* $\mathcal{E}_{heart} \cup \big(\cup_{v\in\Lambda(G)}\mathcal{F}(v) - \{v\}\big) \cup \big(\cup_{v\in\Pi(G)}\mathcal{F}(v)\big)$.

Example 5. *Consider the stopping graph $G = (V \cup C, E)$ in Fig. 1. When we run Algorithm $OPT_\infty(G)$, since G is not smooth, we need to find an optimal elimination set for its heart-component $\mathcal{C}_G$ (the part shown in a dashed circle in the figure) using Step 2, which is found by making another call to the algorithm. (That recursive call will in turn find an optimal elimination set for the heart-component of $\mathcal{C}_G$, which consists of a path of three nodes and another isolated node.) The optimal elimination sets for bridge-trees are found using the $Tree_\infty$ Algorithm.* □

We now analyze the algorithm $OPT_\infty(G)$, and prove that it returns an optimal elimination set for $G = (V \cup C, E)$.

First, let $G_\alpha = (V_\alpha \cup C_\alpha, E_\alpha)$ denote the subgraph of G obtained by removing all peripheral trees rooted at nodes in $\cup_{v\in\Lambda(G)}\mathcal{A}_G(v)$, namely, peripheral-trees adjacent to bridge-variable-nodes. (Here $V_\alpha \subseteq V$ and $C_\alpha \subseteq C$.)

Lemma 6. *Let $G = (V \cup C, E)$ be a stopping graph. For the SSE_∞ Problem, there exists an optimal elimination set $\mathcal{E}^* \subseteq V$ such that $\mathcal{E}^* \cap V_\alpha$ is an elimination set for G_α.*

Proof: Let $\hat{S} \subseteq V$ be an optimal elimination set for G. Consider a bridge-variable-node $v \in \Lambda(G)$. Let $G_\gamma(v) = (V_\gamma(v) \cup C_\gamma(v), E_\gamma(v))$ denote the subgraph of G obtained by removing the peripheral-trees adjacent to v. Consider the bridge-tree $\mathcal{T}_{bt,G}(v) = (V_{bt,G}(v) \cup C_{bt,G}(v), E_{bt,G}(v))$. There are two possible cases.

CASE 1: $\hat{S} \cap V_{bt,G}(v)$ is an elimination set for $\mathcal{T}_{bt,G}(v)$.

In this case, by the property of the $Tree_\infty$ Algorithm mentioned earlier, there exists an optimal elimination set $P \subseteq V_{bt,G}(v)$ for $\mathcal{T}_{bt,G}(v)$ that contains the root v. So $v \in P$ and $|P| \le |\hat{S} \cap V_{bt,G}(v)|$. Consider the set $S^* \triangleq (\hat{S} - V_{bt,G}(v)) \cup P$. Clearly, $|S^*| \le |\hat{S}|$. By removing the nodes in S^* from G, all the variable nodes in $\mathcal{T}_{bt,G}(v)$ become decodable (because $P \subseteq S^*$ is an elimination set for $\mathcal{T}_{bt,G}(v)$), and all the variable nodes in G but not in $\mathcal{T}_{bt,G}(v)$ also become decodable (because the bridge-variable-node v is the only connection between $\mathcal{T}_{bt,G}(v)$ and the rest of G, so when v is removed, it only helps decode the variable nodes in $V - V_{bt,G}(v)$). So S^* must be an optimal elimination set for G. Since $v \in S^*$, $S^* \cap V_\gamma(v)$ must be an elimination set for $G_\gamma(v)$ because once the bridge-variable-node v is removed, the decoding of nodes in $G_\gamma(v)$ has got all the help it possibly could from the bridge-tree $\mathcal{T}_{bt,G}(v)$.

CASE 2: $\hat{S} \cap V_{bt,G}(v)$ is not an elimination set for $\mathcal{T}_{bt,G}(v)$.

In this case, since v is the only connection between $\mathcal{T}_{bt,G}(v)$ and $G_\gamma(v)$, after nodes in $\hat{S}$ are removed from G, the BP decoding must recover the value of v by decoding nodes in $G_\gamma(v)$ (not by decoding nodes in $\mathcal{T}_{bt,G}(v)$). So $\hat{S} \cap V_\gamma(v)$ must be an elimination set for $G_\gamma(v)$.

So in both cases, G has an optimal elimination set $S \subseteq V$ (which is either S^* or $\hat{S}$ in the above cases) such that $S \cap V_\gamma(v)$ is an elimination set for $G_\gamma(v)$. The above analysis handles a single bridge-variable-node. It it not difficult to see that we can handle all the bridge-variable-nodes in the same way, and as a result, we get an optimal elimination set $\mathcal{E} \subseteq V$ such that $\mathcal{E} \cap V_\alpha$ is an elimination set for G_α. So the conclusion holds. ■

Lemma 7. *Let $G = (V \cup C, E)$ be a stopping graph, and let $\mathcal{C}_G = (V_{heart} \cup C_{heart}, E_{heart})$ be its heart-component, where $V_{heart} \subseteq V$ and $C_{heart} \subseteq C$. Then for the SSE_∞ Problem, there exists an optimal elimination set $\mathcal{F} \subseteq V$ such that $\mathcal{F} \cap V_{heart}$ is an elimination set for $\mathcal{C}_G$.*

Proof: By Lemma 6, there exists an optimal elimination set $\mathcal{E}^* \subseteq V$ for G such that $\mathcal{E}^* \cap V_\alpha$ is an elimination set for G_α. (Note that $\mathcal{C}_G$ is a subgraph of G_α and can be obtained from G_α by removing all the trees in $\{\mathcal{T}_{bt,G}(v) \mid v \in \Pi(G)\}$.)

$\forall\ v \in \Pi(G)$, let $\mathcal{B}_G(v) \subseteq \mathcal{A}_G(v)$ be the set of nodes with this property: for every node $u \in \mathcal{B}_G(v)$, after nodes in $\mathcal{E}^*$ are removed from G, if we run the BP decoding algorithm only on the tree $\mathcal{T}_{pt,G}(u)$, the variable nodes in $\mathcal{T}_{pt,G}(u)$ would not be fully decoded (which is equivalent to saying that the root u would not be decoded, because nodes outside the tree can help decode the tree only through u). Then, by the property of the $Tree_\infty$ Algorithm, we can require $\mathcal{E}^*$ to have this property: $\forall\ u \in \mathcal{A}_G(v) - \mathcal{B}_G(v)$, we have $u \in \mathcal{E}^*$.

We now transform $\mathcal{E}^*$ to another optimal elimination set step by step as follows. In each step, consider a bridge-check-node $v \in \Pi(G)$ with $|\mathcal{B}_G(v)| = 0$, and let u be an adjacent variable node in $\in \mathcal{A}_G(v)$. Note that when nodes in $\mathcal{E}^*$ are removed from G and we run the BP decoding algorithm, the check node v must be used to decode one of its neighboring variable node w in the smooth-component $\mathcal{S}_G$, because otherwise we can exclude u from $\mathcal{E}^*$ and use v to decode u instead, and get a smaller elimination set (which is a contradiction). Now to transform $\mathcal{E}^*$, we replace u by w in $\mathcal{E}^*$. After this transformation, the nodes outside $\mathcal{T}_{pt,G}(v)$ clearly

[1] It will be analyzed later how to obtain $\mathcal{E}$ here (or approximate it) based on properties of G to obtain a polynomial-time algorithm.

can still be decoded, and u will be decoded using v after v's other neighboring variable nodes are all decoded. So the transformed $\mathcal{E}^*$ is still an optimal elimination set; yet now we have $|\mathcal{B}_G(v)| = 1$. So after we repeat the above transformation step for every node $v \in \Pi(G)$ with $|\mathcal{B}_G(v)| = 0$, we get an optimal elimination set $\mathcal{E}^*$ with this property: $\forall\ v \in \Pi(G)$, we have $|\mathcal{B}_G(v)| \geq 1$ (actually $|\mathcal{B}_G(v)| = 1$ because otherwise some peripheral-trees adjacent to v will not be decodable). It means that in BP decoding, first all the nodes in the heart-component $\mathcal{C}_G$ can be decoded; and then for every $v \in \Pi(G)$, the unique variable node $u \in \mathcal{B}_G(v)$ can be decoded using check node v; then all peripheral-trees can be decoded. So let $\mathcal{F}$ be the transformed $\mathcal{E}^*$, and the conclusion holds. ■

For any stopping graph G', let $\chi(G')$ denote the size of an optimal elimination set for G'.

Theorem 8. *Let $G = (V \cup C, E)$ be a stopping graph. Then*

$$\chi(G) = \chi(\mathcal{C}_G) - |\Lambda(G)| + \sum_{v \in \Lambda(G) \cup \Pi(G)} \chi(\mathcal{T}_{bt,G}(v)).$$

Proof: Let $\mathcal{C}_G = (V_{heart} \cup C_{heart}, E_{heart})$ be the heart-component of G. By Lemma 7, there exists an optimal elimination set $\mathcal{F}$ for G such that $\mathcal{F} \cap V_{heart}$ is an elimination set for $\mathcal{C}_G$. So $|\mathcal{F}| = \chi(G)$ and $|\mathcal{F} \cap V_{heart}| \geq \chi(\mathcal{C}_G)$.

In any an elimination set for G, to decode any bridge-tree $\mathcal{T}_{bt,G}(v)$ rooted at a bridge-variable-node $v \in \Lambda(G)$, it is necessary to remove at least $\chi(\mathcal{T}_{bt,G}(v)) - 1$ non-root nodes (which, together with v, form an elimination set for $\mathcal{T}_{bt,G}(v)$). To decode any bridge-tree $\mathcal{T}_{bt,G}(v)$ rooted at a bridge-check-node $v \in \Pi(G)$, the nodes removed in $\mathcal{T}_{bt,G}(v)$ needs to be an elimination set for $\mathcal{T}_{bt,G}(v)$, because other nodes in G connected to $\mathcal{T}_{bt,G}(v)$ are all connected to the check node v, which makes it no easier to decode $\mathcal{T}_{bt,G}(v)$ than running BP decoding on $\mathcal{T}_{bt,G}(v)$ alone. So $\chi(G) = |\mathcal{F}| \geq |\mathcal{F} \cap V_{heart}| + \sum_{v \in \Lambda(G)} (\chi(\mathcal{T}_{bt,G}(v)) - 1) + \sum_{v \in \Pi(G)} \chi(\mathcal{T}_{bt,G}(v)) \geq \chi(\mathcal{C}_G) - |\Lambda(G)| + \sum_{v \in \Lambda(G) \cup \Pi(G)} \chi(\mathcal{T}_{bt,G}(v))$. It is not hard to see that an elimination set whose size matches the above lower bound can be constructed. So the conclusion holds. ■

Theorem 9. *The algorithm $OPT_\infty(G)$ returns an optimal elimination set for $G = (V \cup C, E)$.*

Proof: The proof is by induction. Note that $OPT_\infty(G)$ is a recursive algorithm. As the base case, if G is smooth, which means the heart-component $\mathcal{C}(G)$ is the same as G (including the special case that $\mathcal{C}(G)$ is an empty graph), the algorithm will return an optimal elimination set in its Step 1. Now as the inductive step, assume G is not smooth. The algorithm (in its Step 2 through Step 4) constructs and returns an elimination set whose size matches the optimal size shown in Theorem 8, whose BP decoding process is as described in the end of the proof of Lemma 7 (for the transformed elimination set $\mathcal{E}^*$). That completes the induction; so the conclusion holds. ■

B. Efficient Algorithm for Stopping Graph with One Cycle

The algorithm $OPT_\infty(G)$ finds an optimal elimination set, but it may not be a polynomial-time algorithm, because it is NP-hard to find an optimal elimination set for a smooth stopping graph (which can be derived from the NP-hardness of the SSE_∞ Problem). In this subsection, we study the SSE_∞ Problem for stopping graphs containing just one cycle, which is the basic case for cyclic stopping graphs, and show that an efficient algorithm exists.

For a stopping graph $G = (V \cup C, E)$ that contains one cycle, its heart-component $\mathcal{C}_G$ is either a cycle (if $|\Pi(G)| = 0$) or $|\Pi(G)|$ disjoint paths (if $|\Pi(G)| > 0$). If $\mathcal{C}_G$ is a cycle, then its optimal elimination set contains a single variable node in the cycle. If $\mathcal{C}_G$ consists of $|\Pi(G)|$ disjoint paths, then its optimal elimination set contains $|\Pi(G)|$ variable nodes (one for each path). So based on Theorem 8, we see that an optimal elimination set for G has size

$$\chi(G) = \max\{|\Pi(G)|, 1\} - |\Lambda(G)| + \sum_{v \in \Lambda(G) \cup \Pi(G)} \chi(\mathcal{T}_{bt,G}(v)).$$

Based on the above analysis, the algorithm $OPT_\infty(G)$ can be simplified accordingly. (We skip details of the algorithm due to space limit.) Note that the $Tree_\infty$ Algorithm has linear time complexity [3]. It is not hard to see that the algorithm here also has linear time complexity $O(V + C + E)$.

C. Approximation Algorithm for p-Cyclic Stopping Graph

We now present an approximation algorithm for p-cyclic stopping graphs. It uses an approximation algorithm for the Minimum Feedback Vertex Set (MFVS) Problem as a tool. Given an undirected graph $G' = (V', E')$, a *feedback vertex set* (FVS) is a set of vertices $S \subseteq V'$ such that every cycle in G' contains at least one vertex of S (namely, removing S will turn G' into an acyclic graph). The MFVS Problem is defined as follows: given an undirected graph $G' = (V', E')$ where every vertex $v \in V'$ has a non-negative cost $c(v)$, find an FVS in G' whose cost is minimized. The MFVS Problem has a 2-approximation algorithm [1], which we shall call the *MFVS Algorithm*.

Our approximation algorithm for the SSE_∞ Problem for a p-cyclic stopping graph $G = (V \cup C, E)$ is a modification of the Algorithm $OPT_\infty(G)$. In Step 1 of $OPT_\infty(G)$, instead of finding an optimal elimination set $\mathcal{E} \subseteq V$ for the smooth graph G, we find an approximate solution as follows:

- Step 1.1: Let c be a non-negative integer. Use exhaustive search to check if G has an elimination set of size at most c. If yes, return an optimal elimination set for G; otherwise, go to Step 1.2.
- Step 1.2: In G, let all variable nodes have cost 1, and let all check nodes have cost ∞. Run the MFVS Algorithm to find an FVS $F \subseteq V$ for G. (The high cost for check nodes ensures that F contains only variable nodes.)
- Step 1.3: Remove the nodes in F from G, and use BP decoding to further remove those nodes that become decodable until we get a new stopping graph $\hat{G} = (\hat{V} \cup \hat{C}, \hat{E})$. ($\hat{G}$ is acyclic because F is an FVS.) Then find an optimal elimination set $S^* \subseteq \hat{V}$ for $\hat{G}$ using the $Tree_\infty$ Algorithm. Return $F \cup S^*$ as the elimination set for G.

We let Step 2 through Step 4 of the algorithm $OPT_\infty(G)$ remain the same. Let us call the new algorithm $Approx_\infty(G)$. It has polynomial time complexity for constant parameter c because all its elements (the MFVS Algorithm, the $Tree_\infty$ Algorithm, the exhaustive search, the number of recursive calls) are of polynomial time. We now analyze its approximation ratio (define based on the size of the elimination set).

Theorem 10. *Algorithm $Approx_\infty(G)$ has approximation ratio*

$$\frac{2p}{c+1}+1.$$

Proof: It is sufficient to analyze the approximation ratio for the elimination set found by Step 1.1 through Step 1.3 of the algorithm, because the elimination sets found in the other steps are optimal. If the smooth graph G has an elimination set of size at most c, an optimal elimination set for G is returned.

Now consider the case that G has no elimination set of size at most c, which means $\chi(G) \geq c+1$. In Step 1.2, we have $|F| \leq 2p$ (because the smooth graph G here is also p-cyclic, and the MFVS Algorithm has approximation ratio 2). In Step 1.3, we have $|S^*| = \chi(\hat{G}) \leq \chi(G)$ (because having some variable nodes removed will only be helpful for decoding the remaining variable nodes). So the ratio $\frac{|F \cup S^*|}{\chi(G)} = \frac{|F|+|S^*|}{\chi(G)} \leq \frac{2p+\chi(G)}{\chi(G)} = \frac{2p}{\chi(G)} + 1 \leq \frac{2p}{c+1} + 1$. ∎

So when p is small, the approximation ratio can be small.

III. SSE_k Problem for p-Cyclic Stopping Graphs and Dense Stopping Graphs

A. Approximation Algorithm for p-Cyclic Stopping Graph

The SSE_k Problem is a generalization of the SSE_∞ Problem. We present an approximation algorithm for it for a p-Cyclic Stopping Graph $G = (V \cup C, E)$. It is the same as Step 1.1 through Step 1.3 of the Algorithm $Approx_\infty(G)$, except that we use the $Tree_k$ Algorithm to replace the $Tree_\infty$ Algorithm in Step 1.3; and we do not use the remaining steps (namely Step 2 through Step 4 of $OPT_\infty(G)$). (That is because here G is the original input to the algorithm, not its smooth subgraph obtained during the recursion.) Let us call the new algorithm $Approx_k(G)$.

Algorithm $Approx_k(G)$ also has an approximation ratio of $\frac{2p}{c+1}+1$ and polynomial time complexity. The analysis is very similar to Algorithm $Approx_\infty(G)$, so we skip it details. Note that although the two algorithms have the same approximation ratio (which considers the worst case performance) and Algorithm $Approx_k(G)$ appears simpler (i.e., with fewer steps), Algorithm $Approx_\infty(G)$ is better optimized for the SSE_∞ Problem because its extra steps can further reduce the size of the output elimination set.

B. Approximation Algorithm for Dense Stopping Graphs

In this subsection, we analyze how an important factor, RBER (raw bit-erasure rate), affects the performance of approximation algorithms. We show that for high-rate codes with high actual erasure rates (which leads to dense stopping graphs), all algorithms have good approximation ratios.

Consider an (N, K) LDPC code with N codeword bits and K information bits (where $K < N$), whose code rate is $R \triangleq K/N$. Let $G = (V \cup C, E)$ be its Stopping Graph, where V is the Stopping Set. It is simple to show that the higher RBER is, the greater $|V|$ becomes on average. Let $\epsilon \triangleq |V|/N$ be called the *actual erasure rate relative to stopping set* V.

Lemma 11. *Let $S \subseteq V$ be any solution (i.e., an Elimination Set) to the SSE_k Problem. If $|V| \geq N - K$, then*

$$|S| \geq |V| - N + K.$$

Proof: The proof is by contradiction. If $|S| < |V| - N + K$, then after the erased bits in the Elimination Set are removed, the total number of codeword bits with known values is $(N - |V|) + |S| < (N - |V|) + (|V| - N + K) = K$. Then the BP decoder will not be able to recover the K bits of information in the codeword. ∎

Theorem 12. *For the SSE_k Problem, if $\epsilon > 1 - R$, the approximation ratio of any algorithm is at most*

$$\frac{\epsilon}{\epsilon - (1-R)}.$$

Proof: Let S^* and S be an optimal solution and the solution of an arbitrary algorithm, respectively, to the SSE_k Problem. If $\epsilon > 1 - R$, then $|V| = \epsilon N > (1 - K/N)N = N - K$. By Lemma 11, $|S^*| \geq |V| - N + K$. Since $S \subseteq V$, we get $\frac{|S|}{|S^*|} \leq \frac{|V|}{|V|-N+K} = \frac{\epsilon}{\epsilon-1+R}$. ∎

So for high rate codes (where R approaches 1), if the RBER is high (which approaches 1), then with high probability, ϵ also approaches 1. In this case, $\frac{\epsilon}{\epsilon-(1-R)}$ approaches 1, so all algorithms have good approximation ratios.

IV. Conclusions

This paper studies the stopping set elimination problem for LDPC decoding, which has various applications to data storage and communication. It focuses on p-cyclic stopping graphs, and presents a number of algorithms. The work can be extended by studying more specific LDPC code constructions (e.g., spatially-coupled codes, etc.), and design corresponding SSE_k algorithms.

Acknowledgment: This work was supported in part by NSF Grant CCF-1718886.

References

[1] V. Bafna, P. Berman and T. Fujito, "A 2-approximation algorithm for the undirected feedback vertex set problem," in *SIAM J. Discret. Math.*, vol. 12, no. 3, pp. 289-297, 1999.

[2] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," in *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, 2002.

[3] A. Jiang, P. Upadhyaya, Y. Wang, K. R. Narayanan, H. Zhou, J. Sima and J. Bruck, "Stopping set elimination for LDPC codes," in *Proc. 55th Allerton Conference on Communication, Control and Computing*, 2017. Available at $http://faculty.cse.tamu.edu/ajiang/Publications/2017/StoppingSetElimination\_Allerton.pdf$.

[4] A. McGregor and O. Milenkovic, "On the hardness of approximating stopping and trapping sets," in *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1640–1650, 2010.