Stopping Set Elimination for LDPC Codes

Anxiao (Andrew) Jiang*, Pulakesh Upadhyaya*, Ying Wang*, Krishna R. Narayanan*
Hongchao Zhou†, Jin Sima‡, and Jehoshua Bruck‡

* Computer Science and Engineering Department, Texas A&M University
* Electrical and Computer Engineering Department, Texas A&M University
† School of Information Science and Engineering, Shandong University
‡ Electrical Engineering Department, California Institute of Technology

Abstract—This work studies the Stopping-Set Elimination Problem, namely, given a stopping set, how to remove the fewest erasures so that the remaining erasures can be decoded by belief propagation in $k$ iterations (including $k = \infty$). The NP-hardness of the problem is proven. An approximation algorithm is presented for general $k$ when the stopping sets form trees.

I. INTRODUCTION

In this paper, we study a basic theoretical problem for LDPC codes: when the erasures in a noisy LDPC codeword cannot be corrected by the decoder, how to remove the fewest erasures so that the remaining erasures become decodable? The problem has several applications:

- Distributed Storage. Distributed file systems like HDFS have been widely used in big data applications. Typically, they store data in blocks, and ECCs are applied over the blocks (where each block is seen as a codeword symbol of the ECC). Binary LDPC codes are naturally an attractive candidate for distributed storage, as they have excellent code rates, good locality (e.g., a missing block can be recovered by a local disk from a few neighboring blocks), and excellent computational simplicity (only XOR is used for decoding, since when each block has $t$ bits, the decoding can be seen as $t$ binary LDPC codes being decoded in parallel). Meanwhile, almost all big IT companies store multiple copies of their data at different locations. So when one site loses some blocks in an LDPC code and cannot recover them by itself, it needs to retrieve some lost blocks from other remote sites. Since communication with remote sites is much more costly than accessing local disks, it is desirable to minimize the number of blocks retrieved from remote sites as long as the remaining erasures become decodable.

- Satellite-to-Ground Communication with Feedback. Consider satellite-to-ground communication, where data (e.g., big sensing images) are partitioned into packets (i.e., blocks), and LDPC codes are applied over the packets (similar to the case for distributed storage). As the channel is noisy, some packets received by the ground may be un-decodable, and the ground will request the satellite to retransmit some of those lost packets. Since the satellite-to-ground communication can be costly, it is desirable to minimize the number of retransmitted packets.

Let us define the problem more specifically. Let the LDPC code’s decoder be the following widely-used iterative belief-propagation (BP) algorithm: in each iteration, use every parity-check equation involving exactly one erasure to decode that erasure; and repeat until every equation involves zero or at least two erasures. If the decoding fails, then we are left with a stopping set, which is a set of erasures such that every parity-check equation involving any of them involves at least two of them. If we represent the LDPC code by a bipartite Tanner graph, then a stopping set is a subset of variable nodes (representing erasures) such that a check node adjacent to any of them is adjacent to at least two of them.

We illustrate the average sizes of Stopping Sets for different raw bit-erasure rates (RBERs) in Fig. 1. It is for an (8192,7561) LDPC code of rate 0.923 and regular degrees $(d_v = 3, d_c = 39).$ (For RBERs near the code’s decoding threshold, the uncorrectable bit-erasure rates (UBER) after BP decoding is shown in Fig. 1 (a).) For RBERs in the full range from 0 to 1, the average stopping-set sizes (namely, average number of un-decodable erasures after BP-decoding) are shown in Fig. 1 (b). It can be seen that the average stopping-set size increases approximately linearly (from 0 to 8192) as RBER increases from 0 to 1.

![Fig. 1. Statistics of an (8192,7561) LDPC code. (a) UBER for different RBERs near the code’s decoding threshold. (b) Average stopping-set size for different RBERs.](image-url)
endpoint in $C$. If every node in $C$ has degree two or more, then $G$ is called a Stopping Graph and $V$ is called a Stopping Set. Now let $k \geq 1$ be an integer parameter. If an iterative BP algorithm (as introduced earlier) that runs on $G$ can decode all the variable nodes in $V$ (where every variable node in $V$ is an erasure) within $k$ iterations, then $V$ is called a Decodable Set (or simply decodable); otherwise, it is a Non-Decodable Set (or simply non-decodable). (Here we introduce the parameter $k$ to make the problem more general, and to control not only the decodability of erasures but also the time for decoding.) Note that a Stopping Set must be a Non-Decodable Set, but not vice versa. The problem we study, called Stopping-Set Elimination (SSE$_k$) Problem, is as follows.

**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?

If the constraint on “$k$ iterations” does not exist, we can see $k$ as $\infty$ and call it the SSE$_\infty$ Problem. The rest of the paper is organized as follows. In Section II and III, we prove the NR-hardness of the SSE$_\infty$ Problem and the SSE$_E$ Problem for finite $k$, respectively. In Section IV, we present an approximation algorithm for the latter problem. In Section V, we present efficient algorithms that return optimal solutions for the SSE Problem when the Stopping Sets form tree structures. In Section VI, we present conclusions. Due to space limitation, we skip some details in proofs and analysis. Interested readers can refer to the full paper [3] for the details.

II. NP-HARDNESS OF SSE$_\infty$ PROBLEM

In this section, we prove that the SSE$_\infty$ Problem is NP-hard. The proof has two steps: first, using the well-known Set Cover Problem, we prove that a related covering problem where nearly all elements are covered – which we call the Pseudo Set Cover Problem – is NP-complete; then, we reduce the latter problem to the SSE$_\infty$ Problem.

A. NP-completeness of Pseudo Set Cover Problem

Consider the well-known Set Cover Problem. Let $T = \{t_1, t_2, \ldots, t_n\}$ be a universe of $n$ elements. Let $S_1, S_2, \ldots, S_m$ be $m$ subsets of $T$ such that $T = \bigcup_{i=1}^{m} S_i$. (Each $S_i$ is said to cover its elements.) Let $k \leq m$ be a positive integer. The Set Cover Problem asks if there exist $k$ subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that $T = \bigcup_{j=1}^{k} S_{i_j}$. We now define a Pseudo Set Cover Problem that differs only in its question: it asks if there exist $k$ subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that $|\bigcup_{j=1}^{k} S_{i_j}| \geq |T| - 1$.

**Theorem 2.** The Pseudo Set Cover Problem is NP-complete.

**Proof:** We construct a polynomial-time reduction from the NP-complete Set Cover Problem to the Pseudo Set Cover Problem. Let $T = \{t_1, t_2, \ldots, t_n\}$, $S_1, S_2, \ldots, S_m$, and $k \leq m$ as introduced above. For the corresponding instance of the Pseudo Set Cover Problem, let its universe of elements be $T' = \{t_1, t_2, \ldots, t_n, t_{n+1}\}$, where $t_{n+1}$ is a new element, and let its subsets be $S_1, S_2, \ldots, S_m, S_{m+1}$, where $S_{m+1} = \{t_{n+1}\}$. It is simple to see that the mapping between the two instances takes polynomial (in fact, linear) time.

Let us now prove the following claim: the Set Cover Problem has $k$ subsets covering all the $n$ elements in $T$ if and only if the Pseudo Set Cover Problem has $k$ subsets covering at least $|T'| - 1 = n$ elements in $T'$.

Consider one direction of the proof: suppose that the Set Cover Problem has $k$ subsets covering all elements of $T$. Then the same $k$ subsets cover exactly $n$ elements of $T'$. (The only uncovered element is $t_{n+1}$.)

Now consider the other direction of the proof: suppose that the Pseudo Set Cover Problem has $k$ subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ covering at least $n$ elements in $T'$. Without loss of generality (WLOG), assume that $i_1 < i_2 < \cdots < i_k$. There are three possible cases:

- **Case 1:** The $k$ subsets cover all the $n + 1$ elements of $T'$. Then $i_k = m + 1$, and the remaining $k - 1$ subsets cover all the elements in $T$. By adding to the $k - 1$ remaining subsets any other subset in $\{S_1, S_2, \ldots, S_m\}$, we get $k$ subsets covering all elements of $T$ for the Set Cover Problem.

- **Case 2:** The $k$ subsets cover $n$ elements of $T'$, including $t_{n+1}$. Then $i_k = m + 1$, and there must be exactly one uncovered element in $T$. Say that uncovered element is $t_i$, and let $S_j$ (where $1 \leq j \leq m$) be any subset that contains $t_j$. (Such a subset $S_j$ must exist.) By replacing $S_i = S_{m+1}$ by $S_j$, we get $k$ subsets that cover all the elements of $T$.

- **Case 3:** The $k$ subsets cover $n$ elements of $T'$, but not covering $t_{n+1}$. Then the same $k$ subsets cover all the elements of $T'$.

So the reduction exists, and the conclusion holds.

B. NP-hardness of SSE$_\infty$ Problem

We now prove the NP-hardness of the SSE$_\infty$ Problem by using a reduction from the Pseudo Set Cover Problem. Let us begin with some constructions.

Consider the bipartite graph shown in Fig. 2 (a). It consists of four variable nodes ($s_i, t_j, u_{i,j}$ and $w_{i,j}$) and three check nodes ($c_{i,j}^1, c_{i,j}^2$ and $c_{i,j}^3$). We denote it by $D_{i,j}$ to indicate that it connects node $s_i$ and node $t_j$. We prove some basic property it has on iterative BP decoding.

**Lemma 3.** In the graph $D_{i,j}$ that contains the variable nodes $s_i$, $t_j$, $u_{i,j}$, $w_{i,j}$ as a Stopping Set, if the value of the variable node $s_i$ becomes known, the BP decoding algorithm will recover the values of all the three remaining variable nodes.

**Proof:** If the value of $s_i$ becomes known, by using the check nodes $c_{i,j}^1$ and $c_{i,j}^2$, the BP decoding algorithm will
recover the values of \( u_{i,j} \) and \( w_{i,j} \), respectively. Then via the
check node \( c_{i,j} \), it will recover the value of \( t_j \).

If the value of \( t_j \) becomes known, since \( c_{i,j} \) has degree 3, the
BP algorithm will not recover any more values. \( \blacksquare \)

The graph \( D_{i,j} \) will be viewed as a “gadget” that connects
node \( s_i \) with node \( t_j \). To simplify the presentation, in the
following, we often represent it by the symbol shown in Fig. 2
(b), where the “gate” \( g_{i,j} \) represents the five nodes \( \{c_{i,j}^1, c_{i,j}^2, c_{i,j}^3, u_{i,j}, w_{i,j}\} \) and their incident edges. The “direction” of the
gate \( g_{i,j} \) indicates the “directed” property shown in the above
lemma: decoding \( s_i \) leads to decoding \( t_j \), but not vice versa.

Consider the Pseudo Set Cover Problem with input parameters
\( T = \{t_1, t_2, \ldots, t_n\}, S_1, S_2, \ldots, S_m \) and \( k \leq m \) as
introduced earlier. To reduce it to the \( SSE_\infty \) Problem, we
will map every instance of the Pseudo Set Cover Problem to some
instance of the \( SSE_\infty \) Problem.

Let us start by building a bipartite graph \( G_1 \). We start by
assigning \( m+n \) nodes: for every subset \( S_i \) (for \( 1 \leq i \leq m \)) or
element \( t_j \) (for \( 1 \leq j \leq n \)) in the Pseudo Set Cover Problem,
there is a corresponding variable node \( s_i \) or \( t_j \) in \( G_1 \). Then,
whenever the Pseudo Set Cover Problem has \( t_j \in S_i \), we
connect nodes \( s_i \) and \( t_j \) by the bipartite graph \( D_{i,j} \). The graph
obtained this way is \( G_1 \). An example is shown below.

**Example 4.** Let the Pseudo Set Cover Problem be \( T = \{t_1, t_2, t_3, t_4, t_5\} \) and \( S_1 = \{t_1, t_3, t_4\}, S_2 = \{t_1, t_3\}, S_3 = \{t_2, t_4, t_5\}. \) (As \( k \) is irrelevant to the mapping, we do not specify it.) It is illustrated in Fig. 2 (c), where there is an edge between
\( S_i \) and \( t_j \) if and only if \( t_j \in S_i \). The corresponding graph \( G_1 \)
is shown in both Fig. 2 (d) and (e), where the symbol for each

\( D_{i,j} \) is used in Fig. 2 (d), and the full details of \( G_1 \) are shown
in Fig. 2 (e). It is easy to see the correspondence between \( G_1 \)
and the Pseudo Set Cover Problem. \( \Box \)

We now create a bipartite graph \( G_{11} \) as follows. Given graph
\( G_1 \), we add \( m+1 \) additional check nodes \( c_0, c_1, c_2, \ldots, c_m. \)
For \( 0 \leq i \leq m \) and \( 1 \leq j \leq n \), add an edge between
the check node \( c_i \) and the variable node \( t_j \). For \( 1 \leq i \leq m \), add
an edge between the check node \( c_i \) and the variable node \( s_i \).
The graph obtained this way is \( G_{11} \). (For example, following
Example 4, \( G_{11} \) is as shown in Fig. 2 (f).)

In the following, we consider only cases where \( n > 1 \). (The
case \( n = 1 \) is trivial.) It is then simple to see that in \( G_{11} \), the
degree of every check node is at least two. So it is a Stopping
Graph, namely, an instance of the \( SSE_\infty \) Problem.

**Lemma 5.** If for the Pseudo Set Cover Problem, there exist \( k \) subsets
that cover at least \( n-1 \) elements of \( T \), then for the corresponding graph \( G_{11} \), \( k \) variable nodes can be removed so
that the remaining variable nodes form a Decodable Set.

**Proof:** Suppose that \( S_{i_1}, S_{i_2}, \ldots, S_{i_k} \) are \( k \) chosen subsets
that cover at least \( n-1 \) elements of \( T \). Let us remove
the corresponding \( k \) variable nodes \( s_{i_1}, s_{i_2}, \ldots, s_{i_k} \) from
the graph \( G_{11} \). Since removing a variable node is equivalent to
turning the node from an erasure to a known value, by the
“directed” property of \( D_{i,j} \) proved earlier, we know that the
BP decoding algorithm will recover the values of at least
\( n-1 \) variable nodes among \( t_1, t_2, \ldots, t_n \). That is because
if an element \( t_j \) is covered by some chosen subset \( S_{i_r} \) (where
\( 1 \leq r \leq k \)), since the value of the variable node \( s_{i_r} \) is now
known, via the “gadget” \( D_{i_r,j} \), the BP decoding algorithm can
recover the value of \( t_j \).

We now show that the BP decoding algorithm can recover
the values of all \( n \) variable nodes \( t_1, t_2, \ldots, t_n \). From the
above discussion, we know that at most one of them – say \( t_x \)
– is not decoded yet. So the BP algorithm can use the check
node \( c_0 \) (which has degree \( n \)) to recover the value of \( t_x \) as
\( t_x = \oplus_{1 \leq i \leq n, i \neq x} t_i \).

Since the values of \( t_1, t_2, \ldots, t_n \) are all known now, for
\( i = 1, 2, \ldots, m \), the BP decoding algorithm can use the check
node \( c_i \) to recover the value of \( s_i \) (if its value is not already
known). So all the variable nodes can recover their values.
Therefore, the remaining variable nodes form a Decodable Set.

When a set of variable nodes \( S \subseteq V \) is removed from a
Stopping Graph \( G = (V \cup C, E) \), if the remaining nodes of \( V \)
become decodable, we call \( S \) an Elimination Set of size \( |S| \).

**Lemma 6.** If \( G_{11} \) has an Elimination Set of size \( k \leq m \), then
\( G_{11} \) has an Elimination Set of size \( k \) that is also a subset of
\( \{s_1, s_2, \ldots, s_m\} \).

**Proof:** Let \( X = \{x_1, x_2, \ldots, x_k\} \) be an Elimination Set
of \( G_{11} \), where each \( x_i \) is a variable node. Let us create a
set \( Y = \{y_1, y_2, \ldots, y_k\} \subseteq \{s_1, \ldots, s_m\} \) as follows. For
\( i = 1, 2, \ldots, k \), do:
Let us consider its decision problem: given a Stopping Graph

**Theorem 9.**

**Lemma 7.** If $G_{11}$ has an Elimination Set of size $k$ 

\[
\{ s_{i_1}, s_{i_2}, \ldots, s_{i_k}\} \subseteq \{ s_1, s_2, \ldots, s_m\},
\]

then for the corresponding Pseudo Set Cover Problem, the $k$ subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ cover at least $n - 1$ elements of $T$.

**Proof:** The proof is by contradiction. Suppose that $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ cover at most $n - 2$ elements of $T$. Then in $G_{11}$, when the values of \{ $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ \} are known, the BP algorithm can use the “gadgets” $D_{i,j}$ to decode at most $n - 2$ variable nodes among $t_1, t_2, \ldots, t_n$. Then the BP algorithm gets stuck because it cannot use any check node to decode any more variable node:

- For any check node $c_i$ (where $0 \leq i \leq m$), at least two adjacent nodes in \{ $t_1, t_2, \ldots, t_n$ \} are not decoded yet. So the BP algorithm cannot use $c_i$ to decode more variable nodes.
- For any “gadget” $D_{i,j}$ that connects $s_i$ and $t_j$, if $s_i \notin \{ s_{i_1}, s_{i_2}, \ldots, s_{i_k} \}$, by the “directed” property of the gadget, the BP algorithm cannot use it to decode $s_i$ whether the node $t_j$ has been decoded or not.

That means $\{ s_{i_1}, s_{i_2}, \ldots, s_{i_k} \}$ is not an Elimination Set, which is a contradiction. That leads to the conclusion.

By combining the above two lemmas, we get:

**Lemma 8.** If $G_{11}$ has an Elimination Set of size $k \leq m$, then for the corresponding Pseudo Set Cover Problem, there exist $k$ subsets that cover at least $n - 1$ elements of $T$.

We now prove our main result here.

**Theorem 9.** The $SSE_{\infty}$ Problem is NP-hard.

**Proof:** The $SSE_{\infty}$ Problem is an optimization problem. Let us consider its decision problem: given a Stopping Graph $G = (V \cup C, E)$ and a positive integer $k$, does it have an Elimination Set of size $k$? Let us call this decision problem $P_{sse}$. It is clear that $P_{sse} \in NP$.

We have shown a mapping that maps every instance of the Pseudo Set Cover Problem to an instance of $P_{sse}$. The mapping takes polynomial time. By combining Lemma 5 and Lemma 8, we see that the answer to the Pseudo Set Cover Problem is “yes” (namely, there exist $k$ subsets that cover at least $n - 1$ elements of $T$) if and only if the answer to $P_{sse}$ is “yes” (namely, $G_{11}$ has an Elimination Set of size $k$). So the mapping is a polynomial-time reduction. By Theorem 2, the Pseudo Set Cover Problem is NP-complete. So $P_{sse}$ is NP-complete, which leads to the conclusion.

**III. NP-HARDNESS OF $SSE_k$ PROBLEM FOR FINITE $k$**

We now consider a new question: if $k$ is finite (or even a constant), does the $SSE_k$ problem become polynomial-time solvable? A positive answer seems possible at first sight, because having a small $k$ puts more localized constraints on solutions. For example, if $k = 1$, to correct all remaining erasures in just one iteration, in the subgraph induced by the remaining variable nodes and their adjacent check nodes, every variable node needs to be adjacent to at least one check node of degree one. That is a very local property for the bipartite graph and can possibly make the problem simpler. However, our study below will give a negative answer. We will prove that even the $SSE_1$ Problem is NP-hard.

There have been a number of works on the node-deletion problem [1], [2], [4], [6], which can be generally stated as follows: find the minimum number of vertices to delete from a given graph so that the remaining subgraph satisfies a property $\pi$. They focus on properties that are hereditary on induced subgraphs, namely, whenever a graph $G$ satisfies $\pi$, by deleting nodes from $G$, the remaining subgraphs also satisfies $\pi$. However, the $SSE_k$ Problem is not hereditary, because removing a check node can turn a decodable graph (the desired property) into an un-decodable one.

We now prove the NP-hardness of the $SSE_1$ Problem. We use a reduction from the NP-complete Not-all-equal SAT Problem [5], defined as follows: let $x_1, x_2, \ldots, x_n$ be $n$ Boolean variables. A literal is either $x_i$ or $\bar{x}_i$ (namely, the NOT of $x_i$) for some $i \in \{1,2,\ldots,n\}$. Let a clause be a set of three literals. Let $S = \{C_1, C_2, \ldots, C_k\}$ be a set of $k$ clauses. The question is: Is there a truth assignment to the $n$ Boolean variables such that for every clause in $S$, the three literals in the clause are neither all true nor all false (namely, every clause has at least one true literal and also at least one false literal)? (If the answer is “yes”, the problem is called “satisfiable”.)

By convention, “true” is also represented by 1, and “false” is also represented by 0. We give an example of the Not-all-equal SAT Problem.

**Example 10.** Consider the following instance of the Not-all-equal SAT Problem. Let $n = 4$ and $k = 5$. Let the Boolean variables be $x_1, x_2, x_3, x_4$, and let the set of clauses
be \( C_1 = (x_1, \bar{x}_2, x_3), C_2 = (\bar{x}_1, \bar{x}_2, x_4), C_3 = (x_2, x_3, x_4), \)
\( C_4 = (x_1, \bar{x}_3, \bar{x}_4), C_5 = (x_1, x_2, x_3). \) The above instance is satisfiable because we can let the truth assignment be \( x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 1. \) Correspondingly, the clauses become \( C_1 = (1, 0, 0), C_2 = (0, 0, 1), C_3 = (1, 0, 1), \)
\( C_4 = (1, 1, 0), C_5 = (0, 1, 0). \) None of the clauses is \((1, 1, 1)\) (namely, all true) or \((0, 0, 0)\) (namely, all false). \( \square \)

A. Reducing Not-all-equal SAT Problem to SSE\(_3\) Problem

In this subsection, we construct a reduction that maps every instance of the Not-all-equal SAT Problem to an instance of the SSE\(_3\) Problem.

For every Boolean variable \( x_i \) of the Not-all-equal SAT Problem (for \( 1 \leq i \leq n \)), we create a graph as shown in Fig. 3 (a), which will be called the “gadget \( V_i \).” It is a bipartite graph of three variable nodes and three check nodes. (Here nodes \( X_{1i} \) and \( X_{2i} \) represent the true and false values of \( x_i \), respectively.)

For every clause \( C_j \) of the Not-all-equal Problem (for \( 1 \leq j \leq k \)), we create two graphs as shown in Fig. 3 (b), which will be called gadgets \( U_{1j} \) and \( U_{2j} \), respectively. (Here for \( t = 1, 2, 3 \), nodes \( A_{1j}^t \) and \( B_{1j}^t \) represent the true and false values of the \( t \)-th literal in clause \( C_j \), respectively.) We then connect them into one larger gadget \( W_j \) as shown in Fig. 3 (c), where for \( t = 1, 2, 3 \), two paths are used to connect the nodes \( A_{1j}^t \) and \( B_{1j}^t \). (For example, the two paths between \( A_{1j}^1 \) and \( B_{1j}^1 \) have nodes \( d_{1j}^1 \), \( d_{2j}^1 \) and the four check nodes by them.)

In the final graph corresponding to the instance, the gadget \( V_i \) will be connected to the rest of the graph only through nodes \( X_{1i} \) and \( X_{2i} \). So to simplify the presentation, we sometimes represent \( V_i \) by the symbol in Fig. 3 (d), where the two “interface nodes” \( X_{1i} \), \( X_{2i} \) are shown and the remaining details are hidden. Also in the final graph, the gadget \( W_j \) will be connected to the rest of the graph only through nodes \( A_{1j}^1 \), \( A_{1j}^2 \), \( A_{1j}^3 \), \( B_{1j}^1 \), \( B_{1j}^2 \), \( B_{1j}^3 \); so we sometimes represent it by the symbol in Fig. 3 (e).

We now connect the gadgets for clauses to the gadgets for Boolean variables. Consider a clause \( C_j \), and assume its literals are \( C_j = (l_1, l_2, l_3). \) For \( t = 1, 2, 3 \), if \( l_t = x_i \) for some \( 1 \leq i \leq n \), we connect \( A_{1j}^t \) to \( X_{1i} \) and connect \( B_{1j}^t \) to \( X_{2i} \) (through some intermediate nodes) as shown in Fig. 3 (f). Otherwise \( l_t = \bar{x}_i \) for some \( 1 \leq i \leq n \), and we connect \( A_{1j}^t \) to \( X_{1i} \) and connect \( B_{1j}^t \) to \( X_{2i} \) as shown in Fig. 3 (g).

**Example 11.** Assume that a clause is \( C_j = (l_1, l_2, l_3) = (x_1, x_3, \bar{x}_4). \) Its gadget \( W_j \) is connected to the gadgets \( V_1, V_3, V_4 \) as in Fig. 3 (h).

To simplify the presentation of the graph, we represent the connection between a node \( A_{1j}^t \) (or \( B_{1j}^t \)) and a node \( x_i \) (or \( x_i' \)) by a rectangle that is generally denoted by the “H bar.” Then the graph in Fig. 3 (h) is simplified as the presentation in Fig. 3 (i), which shows the connections more clearly. However, it should be noted that each \( A_{1j}^1, B_{1j}^1, x_i \) or \( x_i' \) is connected to an H bar via two edges, not one. \( \square \)

By now, we have constructed the whole graph that corresponds to an instance of the Not-all-equal Problem. The graph will be denoted by \( G_{sse} \). Let us see an example.

---

Fig. 3. (a) The gadget corresponding to a Boolean variable \( x_i \), for \( i = 1, 2, \ldots, n \). (b) Two gadgets corresponding to a clause \( C_j \), for \( j = 1, 2, \ldots, k \). (c) The connected gadget corresponding to a clause \( C_j \), for \( j = 1, 2, \ldots, k \). (d) Symbol for \( V_i \). (e) Symbol for \( W_j \). (f) Connect clause gadget to Boolean variable gadget: case one. (g) Connect clause gadget to Boolean variable gadget: case two. (h) An example of connecting a clause gadget to variable gadgets. (i) Simplified representation of the graph in (h). (j) The graph \( G_{sse} \) corresponding to the Not-all-equal Problem where \( n = 4, k = 2, C_1 = (x_1, x_3, x_4), C_2 = (x_1, x_2, x_3), \) where gadgets are represented by symbols. (k) A Stopping Tree \( G = (V \cup C, E) \). (l) Its BFS (Breadth-First Search) tree \( G_{BFS} \).
Example 12. For the Not-all-equal Problem, let \( n = 4 \) and \( k = 2 \). Let the two clauses be \( C_1 = (x_1, x_3, \bar{x}_4) \), \( C_2 = (x_1, \bar{x}_2, x_3) \). Then the corresponding graph \( G_{sse} \) is shown in Fig. 3 (j), where its gadgets are represented by symbols for clarity. □

It is easy to see that \( G_{sse} \) is a bipartite graph, where every check node has degree more than one. (Specifically, every check node has degree two.) So \( G_{sse} \) is a Stopping Graph.

The subsequent analysis will prove that the Not-all-equal SAT Problem is satisfiable if and only if \( G_{sse} \) has an Elimination Set of size \( n + 3k \) such that after its nodes are removed, the BP algorithm can decode the remaining variable nodes in just one iteration.

B. Properties of Reduction

The mapping from any instance of the Not-all-equal SAT Problem to a graph \( G_{sse} \) has been shown. We now analyze its properties. Due to space limitation, we only present sketches of proofs. For detailed proofs, please see [3].

Let the bipartite graph \( G_{sse} \) be \( G_{sse} = (V_{sse} \cup C_{sse}, E_{sse}) \), where \( V_{sse} \) is the set of variable nodes, \( C_{sse} \) is the set of check nodes, and \( E_{sse} \) is the set of edges. We now define the concepts of Interface Nodes, One-Iteration Elimination Set and Canonical Elimination Set.

Definition 13. Let \( I_{sse} \equiv \{ X_i^1 \mid 1 \leq i \leq n, 0 \leq j \leq 1 \} \cup \{ A_j^1 \mid 1 \leq i \leq k, 1 \leq j \leq 3 \} \cup \{ B_j^1 \mid 1 \leq i \leq k, 1 \leq j \leq 3 \} \) be a subset of variable nodes in \( G_{sse} \). Every node in \( I_{sse} \) is called an “Interface Node.” (As an example, the interface nodes are shown as circles in Fig. 3 (j).)

Definition 14. Let \( T \subseteq V_{sse} \) be a set of variable nodes in \( G_{sse} \). If after removing \( T \) from \( G_{sse} \), the BP algorithm can decode the remaining variable nodes in one iteration, then \( T \) is called a “One-Iteration Elimination Set.”

If \( T \) is a one-iteration elimination set and \( T \subseteq I_{sse} \), then \( T \) is called a “Canonical Elimination Set.”

Lemma 15. If \( G_{sse} \) has a One-Iteration Elimination Set of \( \alpha \) nodes, then \( G_{sse} \) also has a Canonical Elimination Set of at most \( \alpha \) nodes.

Sketches of proof: Let \( F \subseteq V_{sse} \) be a One-Iteration Elimination Set of \( \alpha \) nodes. We will prove the existence of a Canonical Elimination Set \( \hat{F} \subseteq I_{sse} \) with \( |\hat{F}| \leq \alpha \) nodes. Note that the nodes in \( G_{sse} \) are in three kinds of gadgets: gadget \( V_i \), gadget \( W_j \), or \( H \) bar. (See Fig. 3 (j) for an illustration.) The main idea of the proof is to transform \( F \) into \( \hat{F} \) by switching nodes of \( F \) to interface nodes. For example, consider a gadget \( V_i \) (for \( 1 \leq i \leq n \).) (See Fig. 3 (a) for an illustration.) Note that \( X_i^0 \) and \( X_i^0 \) are the only two nodes connecting to nodes outside \( V_i \) in \( G_{sse} \). If \( y_i \in F \), \( X_i^1 \notin F \) and \( X_i^0 \notin F \), we can delete \( y_i \) from \( F \) and still get a one-iteration elimination set, because \( y_i \)'s two neighboring check nodes already have degree 1 after \( X_i^1 \) and \( X_i^0 \) are removed from \( G_{sse} \). If \( y_i \in F \), \( X_i^1 \notin F \) and \( X_i^0 \notin F \), we can replace \( y_i \) by \( X_i^0 \) and still get a one-iteration elimination set, because after \( X_i^1 \) is removed from \( G_{sse} \), both \( X_i^0 \) and \( y_i \) will have a neighboring check node of degree 1, and more edges will be removed in the part of the graph \( G_{sse} \) that is outside \( V_i \). Similar analysis can be applied to other cases of \( V_i \) and to other gadgets (\( W_j \) and \( H \) bar) to show that non-interface nodes in \( F \) can be either deleted or replaced by interface nodes to get \( \hat{F} \). That leads to the conclusion. □

Some properties of Canonical Elimination Sets are shown in the next lemma. We first define “endpoints of an \( H \) bar.”

Definition 16. Let \( u \) be any node in \( \{ A_j^1 \mid 1 \leq j \leq k, 1 \leq t \leq 3 \} \cup \{ B_j^1 \mid 1 \leq j \leq k, 1 \leq t \leq 3 \} \), and let \( v \) be any node in \( \{ X_i^1 \mid 1 \leq i \leq n \} \cup \{ X_i^0 \mid 1 \leq i \leq n \} \). If \( u \) and \( v \) are connected by an \( H \) bar, then they are called the two endpoints of that \( H \) bar.

Example 17. In Fig. 3 (f), the endpoints of \( H \) bars are \( (A_j^1, X_i^1) \) and \( (B_j^1, X_i^0) \). In Fig. 3 (g), such endpoint pairs are \( (A_j^1, X_i^0) \) and \( (B_j^1, X_i^1) \). □

Lemma 18. For the graph \( G_{sse} \), a Canonical Elimination Set \( F \) has the following properties: (1) Property 1: For \( i = 1, \ldots, n \), either \( X_i^1 \in F \) or \( X_i^0 \in F \); (2) Property 2: For \( j = 1, 2, \ldots, k \) and \( t = 1, 2, 3 \), either \( A_j^t \in F \) or \( B_j^t \in F \); (3) Property 3: For \( j = 1, 2, \ldots, k \), \( |F \cap \{ A_j^1, A_j^2, A_j^3 \} | \geq 1 \) and \( |F \cap \{ B_j^1, B_j^2, B_j^3 \} | \geq 1 \); (4) Property 4: If \( u \) and \( v \) are the two endpoints of an \( H \) bar, then either \( u \in F \) or \( v \in F \).

Sketches of proof: For the gadget \( V_i \) (see Fig. 3 (a)), if neither \( X_i^1 \) nor \( X_i^0 \) is in \( F \), then the BP algorithm cannot decode \( y_i \) in one iteration since both of \( y_i \)'s neighboring check nodes will have degree 2. So Property 1 is true. The other three properties can be proved similarly. □

Corollary 19. If \( F \) is a One-Iteration Elimination Set of \( G_{sse} \), then \( |F| \geq n + 3k \).

Proof: If \( F \) is a Canonical Elimination Set, by Property 1 and Property 2 in Lemma 18, we get \( |F| \geq n + 3k \). Then by Lemma 15, the same conclusion holds for any One-Iteration Elimination Set.

Definition 20. Let \( F \) be a Canonical Elimination Set of \( G_{sse} \). If \( |F| = n + 3k \), then \( F \) is called an “Ideal Elimination Set” of \( G_{sse} \). (Here “Ideal” means “of minimum possible size.” Note that an Ideal Eliminate Set may or may not exist for \( G_{sse} \).)

The next lemma easily follows from Lemma 18.

Lemma 21. An Ideal Elimination Set \( F \) of \( G_{sse} \) has these properties: (1) Property 1: For \( i = 1, 2, \ldots, n \), either \( X_i^1 \) or \( X_i^0 \) is in \( F \), but not both; (2) Property 2: For \( j = 1, 2, \ldots, k \) and \( t = 1, 2, 3 \), either \( A_j^t \) or \( B_j^t \) is in \( F \), but not both; (3) Property 3: For \( j = 1, 2, \ldots, k \), in the set \( \{ A_j^1, A_j^2, A_j^3 \} \), at least one node is in \( F \), and at least one node is not in \( F \). The same is true for the set \( \{ B_j^1, B_j^2, B_j^3 \} \); (4) Property 4: If \( u \) and \( v \) are the two endpoints of an \( H \) bar, then either \( u \) or \( v \) is in \( F \), but not both.
Given an Ideal Elimination Set of \( G_{sse} \), we can construct a solution to the Not-all-equal SAT Problem as follows.

**Definition 22.** Let \( F \) be an Ideal Elimination Set of \( G_{sse} \). A corresponding solution \( Sol(F) \) for the Not-all-equal SAT Problem is constructed as follows: \( \forall 1 \leq i \leq n \), the Boolean variable \( x_i = 1 \) (namely, \( x_i \) is true) if and only if \( X^1_i \in F \).

Clearly, in the above solution \( Sol(F) \), a Boolean variable \( x_i = 0 \) (namely, \( x_i \) is false) if and only if \( X^0_i \in F \).

**Lemma 23.** Let \( F \) be an Ideal Elimination Set of \( G_{sse} \), and let \( Sol(F) \) be its corresponding solution to the Not-all-equal SAT Problem. Then for \( 1 \leq j \leq k \) and \( 1 \leq t \leq 3 \), the \( t \)-th literal in the clause \( C_j \) is "true" if and only if \( A^t_j \notin F \).

**Proof:** Let \( l^t_j \) denote the \( t \)-th literal in the clause \( C_j \). Consider two cases:

- Case 1: \( l^t_j \) is \( x_i \) for some \( 1 \leq i \leq n \). In this case, by the construction of \( G_{sse} \), \( A^t_j \) is connected to \( X^1_i \) by an \( H \) bar. \( l^t_j \) is true if and only if \( X^1_i \in F \), which – by Property 4 of Lemma 21 – happens if and only if \( A^t_j \notin F \).
- Case 2: \( l^t_j \) is \( \bar{x}_i \) for some \( 1 \leq i \leq n \). In this case, by the construction of \( G_{sse} \), \( A^t_j \) is connected to \( X^0_i \) by an \( H \) bar. \( l^t_j \) is true if and only if \( x_i \) is false, which happens if and only if \( X^0_i \in F \), which – by Property 4 of Lemma 21 – happens if and only if \( A^t_j \notin F \).

So in both cases, the conclusion holds.

**Lemma 24.** If \( F \) is an Ideal Elimination Set of \( G_{sse} \), then \( Sol(F) \) is a satisfying solution to the Not-all-equal SAT Problem.

**Proof:** For \( 1 \leq j \leq k \), let \( A^1_j \in F \) and \( A^2_j \notin F \). By Property 3 of Lemma 21, such two integers \( t_1, t_2 \in \{1, 2, 3\} \) exist. Consider the clause \( C_j \). By Lemma 23, the \( t_1 \)-th literal of \( C_j \) is false, and \( t_2 \)-th literal of \( C_j \) is true. So for the Not-all-equal SAT Problem, every clause has at least one true literal and at least one false literal. So \( Sol(F) \) is a satisfying solution to the Not-all-equal SAT Problem.

The above lemma is useful for the scenario where \( G_{sse} \) has a One-Iteration Elimination Set of \( n + 3k \) nodes. We now consider another possible scenario: the Not-all-equal SAT Problem is satisfiable.

Given a satisfying solution to the Not-all-equal SAT Problem, we can construct an Ideal Elimination Set of \( G_{sse} \). We first define the corresponding set.

**Definition 25.** Let \( \pi \) be a satisfying solution to the Not-all-equal SAT Problem; that is, with the solution \( \pi \), every clause has at least one true literal and at least one false literal. A corresponding set of nodes, \( F(\pi) \), in \( G_{sse} \) is constructed as follows:

- For \( i = 1, 2, \cdots, n \), if \( x_i = 1 \) in the solution \( \pi \), then \( X^1_i \in F(\pi) \) and \( X^0_i \notin F(\pi) \); otherwise, \( X^1_i \notin F(\pi) \) and \( X^0_i \in F(\pi) \).
- For \( j = 1, 2, \cdots, k \) and \( t = 1, 2, 3 \), if the \( t \)-th literal of clause \( C_j \) is true given the solution \( \pi \), then \( A^t_j \notin F(\pi) \) and \( B^t_j \notin F(\pi) \); otherwise, \( A^t_j \in F(\pi) \) and \( B^t_j \notin F(\pi) \).

**Lemma 26.** Let \( \pi \) be a satisfying solution to the Not-all-equal SAT Problem. Then \( F(\pi) \) is an Ideal Elimination Set of \( G_{sse} \).

**Proof:** We first prove the assertion: \( F(\pi) \) is an One-Iteration Elimination Set of \( G_{sse} \). Consider the gadget \( V_i \), for \( 1 \leq i \leq n \). (See Fig. 3 (a).) By the construction of \( F(\pi) \), either \( X^1_i \) or \( X^0_i \) is in \( F(\pi) \). Either way, after the node in \( F(\pi) \) is removed, the other two variable nodes in \( V_i \) will have neighboring check nodes of degree 1. The other gadgets \( W_j \) and \( H \) can be analyzed similarly. So the assertion holds.

Theorem 27. The SSE1 Problem is NP-hard.

**Sketches of proof:** By Lemmas 15, 24, 26 and Corollary 19, the Not-all-equal SAT Problem is satisfiable if and only if the corresponding SSE1 Problem has a one-iteration elimination set of size \( n + 3k \).

IV. APPROXIMATION ALGORITHM FOR SSE1 PROBLEM

In this section, we present an approximation algorithm for the SSE1 problem, for Stopping Graphs whose degrees of variable nodes and check nodes are upper bounded by \( d_v \) and \( d_c \), respectively. Its approximation ratio is \( d_v(d_c - 1) \). (Clearly, the same result also applies to regular \((d_v, d_c)\) LDPC codes and irregular codes with the same constraint on maximum degrees.) Note that the optimization objective is to minimize the size of the elimination set (namely, the number of removed variable nodes). So the approximation ratio means the maximum ratio of the size of an elimination set produced by the approximation algorithm to the size of an optimal (i.e., minimum) elimination set.

**Definition 28.** In the Stopping Graph \( G = (V \cup C, E) \), \( \forall v \in V \), define its "variable-node neighborhood" as \( \Lambda(v) \triangleq \{ u \in V \setminus \{v\} | \exists c \in C \text{ such that } (u, c) \in E \text{ and } (v, c) \in E \} \). That is, every variable node in \( \Lambda(v) \) shares a common neighboring check node with \( v \).

The algorithm will assign three colors to variable nodes:

- Initially, every variable node is of the color white. It means that this variable node cannot be decoded by one iteration of BP-decoding yet.
- As the algorithm proceeds, if a variable node’s color turns black, it means the algorithm has included it in the Elimination Set (namely, the algorithm has removed it).
- As the algorithm proceeds, if a variable node’s color turns gray, it means the variable node is not yet removed, but it will be decodable after one iteration of BP decoding.

The algorithm works as follows: (1) as initialization, make every variable node white; (2) choose an arbitrary white
variable node \( v \). Let \( U_v \) denote the set of variable nodes in \( \Lambda(v) \) that are currently white or gray; turn the colors of the nodes in \( U_v \) to black, and turn the color of \( v \) to gray. For every check node \( c \) that is connected to at least one variable node in \( U_v \), check if exactly one of \( c \)'s neighboring variable node is white and all \( c \)'s other neighboring variable nodes are black; if so, turn that neighboring variable node’s color from white to gray. (3) repeat the previous step until all variable nodes are either black or gray. Then return the set of black variable nodes as the Elimination Set. The algorithm can be shown to have time complexity \( O(d_v^2 d_c^2 |V|) \).

We now analyze the approximation ratio of the algorithm. Say that the algorithm uses totally \( t \) iterations to identify a sequence of \( t \) white variable nodes \( \tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_t \) in the Stopping Graph \( G = (V \cup C, E) \), and turns the variable nodes in \( U_{\tilde{v}_1}, U_{\tilde{v}_2}, \ldots, U_{\tilde{v}_t} \) black. Let us define a sequence of subgraphs \( G_0, G_1, \ldots, G_t \) accordingly.

**Definition 29.** Let \( G_0 = G \). For \( i = 1, 2, \ldots, t \), let \( G_i \) be obtained from \( G_{i-1} \) by removing the nodes in \( U_{\tilde{v}_i} \cup \{ \tilde{v}_i \} \) and their adjacent edges.

Note that for \( i = 1, 2, \ldots, t \), in the \( i \)-th iteration, the algorithm removes only the variable nodes in \( U_{\tilde{v}_i} \) (namely, turning them black) from the subgraph \( G_{i-1} \), not \( \tilde{v}_i \) or its adjacent check nodes. (It turns \( \tilde{v}_i \) to gray.) However, once \( U_{\tilde{v}_i} \) is removed, all the nodes in \( \Lambda(\tilde{v}_i) \) are removed, so \( \tilde{v}_i \) and its adjacent check nodes become disconnected from the rest of the graph (which is \( G_i \)). Therefore it becomes sufficient to consider the \( SSE_i \) Problem for \( G_i \) in the next iteration, and it can be seen that “\( \tilde{v}_{i+1}, U_{\tilde{v}_{i+1}}, \{ \text{check nodes adjacent to } \tilde{v}_{i+1} \} \), \( \tilde{v}_{i+2}, U_{\tilde{v}_{i+2}}, \{ \text{check nodes adjacent to } \tilde{v}_{i+2} \} \), \ldots, \( \tilde{v}_t, U_{\tilde{v}_t}, \{ \text{check nodes adjacent to } \tilde{v}_t \} \)” are all nodes in \( G_i \).

**Lemma 30.** For \( i = 0, 1, \ldots, t - 1 \), every one-iteration elimination set for \( G_i \) contains at least one variable node in \( U_{\tilde{v}_{i+1}} \cup \{ \tilde{v}_{i+1} \} \).

**Proof:** Consider a one-iteration elimination set \( S \) for \( G_i \). Either \( \tilde{v}_{i+1} \in S \), or \( \tilde{v}_{i+1} \) needs to be decodable in one iteration after \( S \) is removed. The latter requires \( \tilde{v}_{i+1} \) to have a neighboring check node \( c \) such that all \( c \)'s other neighboring variable nodes in \( G_i \) are included in \( S \), and those nodes are all in \( U_{\tilde{v}_{i+1}} \). That leads to the conclusion.

**Lemma 31.** For \( i = 0, 1, \ldots, t \), let \( \alpha_i \) denote the minimum size of a one-iteration elimination set for \( G_i \). Then \( \alpha_i \geq t - i \).

**Proof:** The proof is by induction, but in the reverse order for \( i \) (i.e., from \( i = t, t - 1 \ldots \) down to 0). When \( i = t \), clearly \( \alpha_t \geq t - t = 0 \), so the conclusion holds for the base case. Now assume that the conclusion holds for \( \alpha_{t-i}, \alpha_{t-1}, \ldots, \alpha_{t+1} \), and consider the case for \( \alpha_{i} \).

Consider an optimal (i.e., minimum-sized) one-iteration elimination set \( S \) for \( G_i \). Define \( Y \triangleq S \cap (U_{\tilde{v}_{i+1}} \cup \{ \tilde{v}_{i+1} \}) \). By Lemma 30, \( S \) removes at least one variable node in \( U_{\tilde{v}_{i+1}} \cup \{ \tilde{v}_{i+1} \} \), so \( |Y| \geq 1 \). Let \( \hat{G} \) be the bipartite graph obtained by removing the variable nodes in \( Y \) from \( G_i \) (and their incident edges), and let \( \hat{\alpha} \) denote the minimum size of a one-iteration elimination set for \( \hat{G} \). Then \( \alpha_i = |S| = |Y| + \hat{\alpha} \geq \hat{\alpha} + 1 \).

\( G_{i+1} \) is obtained from \( G_i \) by removing the variable nodes in \( U_{\tilde{v}_{i+1}} \cup \{ \tilde{v}_{i+1} \} \), which is a superset of \( Y \). So \( G_{i+1} \) can also be obtained from \( G \) by removing the variable nodes in \( (U_{\tilde{v}_{i+1}} \cup \{ \tilde{v}_{i+1} \}) \). So \( \alpha_{i+1} \geq \alpha_i \). By the induction assumption, we get \( \alpha_{i+1} \geq t - (i + 1) \). By combining the above results, we get \( \alpha_i \geq \hat{\alpha} + 1 \geq \alpha_{i+1} + 1 \geq t - (i + 1) + 1 = t - i \).

**Theorem 32.** Let \( d_v \) and \( d_c \) denote the maximum degrees of variable nodes and check nodes, respectively, in the Stopping Graph \( G = (V \cup C, E) \). Then the above algorithm has an approximation ratio of \( d_v(d_c - 1) \).

**Proof:** By setting \( i = 0 \) in Lemma 31, we get \( \alpha_0 \geq t \), namely, any one-iteration elimination set for \( G \) removes at least \( t \) variable nodes. The algorithm removes the nodes in \( U_{\tilde{v}_1} \cup U_{\tilde{v}_2} \cup \ldots \cup U_{\tilde{v}_t} \), whose size is \( |U_{\tilde{v}_1}| = \sum_{i=1}^t |U_{\tilde{v}_i}| \leq \sum_{i=1}^t |\Lambda(\tilde{v}_i)| \leq t \cdot d_v(d_c - 1) \). So the approximation ratio is at most \( d_v(d_c - 1) \).

**V. ALGORITHM FOR \( SSE_k \) AND \( SSE_{\infty} \) PROBLEMS**

In this section, we present an algorithms for the \( SSE_k \) Problem for general \( k \geq 1 \), including \( k = \infty \), when the Stopping Graph is a tree (or a forest). The algorithm outputs an optimal solution and has linear time complexity.

The Stopping Graph \( G = (V \cup C, E) \) can be a tree, especially when the RBER is low. In this case, we call \( G \) a Stopping Tree. Note that if \( G \) is a forest, the \( SSE_k \) Problem can be solved for each of its tree components independently.

Given a Stopping Tree \( G = (V \cup C, E) \), we can pick an arbitrary variable node \( v \in V \) as the root, run Breadth-First Search (BFS) on \( G \) starting with \( v \), and label the nodes of \( G \) by \( v_1, v_2, \ldots, v_{|V|+|C|} \) based on their order of discovery in the BFS. (Note that the root node \( v \) is labelled by \( v_1 \), and siblings nodes in the BFS tree always have consecutive labels.) We denote the resulting BFS tree by \( G_{BFS} \).

For any non-root node \( v \) in \( G_{BFS} \), let \( \pi(v) \) denote its parent. Let \( G_{sub} \) denote the subtree of \( G_{BFS} \) obtained this way: if we remove the subtree rooted at \( \pi(v_{|V|+|C|}) \) from \( G_{BFS} \), the remaining subgraph is \( G_{sub} \).

**Example 33.** A Stopping Tree and its BFS tree are shown in Fig. 3 (k) and (l), respectively. (Note that the node labels \( v_1, v_2, \ldots, v_{17} \) in Fig. 3 (k) are not known a priori; instead, they are obtained after we run BFS on the graph with \( v_1 \) as its root.) Here \( v_{|V|+|C|} = v_{17}, \pi(v_{17}) = v_{11}, \) and \( G_{sub} \) is the subtree in the dashed circle in Fig. 3 (l). □

The algorithm first runs BFS on \( G \) to get the tree \( G_{BFS} \) that labels nodes by \( v_1, v_2, \ldots, v_{|V|+|C|} \), where \( v_1 \) is the root. Then it processes the nodes in the reverse order of their labels, and keeps reducing the \( SSE_k \) Problem – actually, a more general form of the \( SSE_k \) Problem, which shall be called the
Definition 34. \([gSSE_k] Problem\) Let \(G = (V \cup C, E)\) be a Stopping Graph and let \(k\) be a non-negative integer. Every variable node \(v \in V\) is associated with two parameters \(\delta(v) \in \{1, 2, \ldots, k, \infty\}\) and \(\omega(v) \in \{0, 1, \ldots, k, \infty\}\) satisfying the condition that either \(\delta(v) = \infty\) or \(\omega(v) = \infty\), but not both; and when the BP decoder runs on \(G\), \(v\)’s value can be recovered (namely, \(v\) can become a non-eraser) by the end of the \(\delta(v)\)-th iteration automatically (namely, without any help from neighboring check nodes). Then, how to remove the minimum number of variable nodes from \(V\) such that for every remaining variable node \(v\) with \(\omega(v) \leq k\), it can be corrected by the BP decoder in no more than \(\omega(v)\) iterations? (By default, if \(\omega(v) = 0\), \(v\) has to be removed from \(V\) because the BP decoder starts with the 1st iteration.)

A solution to the \(gSSE_k\) Problem (namely, the set of removed nodes) is called a \(g\)-Elimination Set. We see that if \(\delta(v) = \infty\) and \(\omega(v) = k\) for every \(v \in V\), then the \(gSSE_k\) Problem is identical to the \(SSE_k\) Problem.

In \(G_{BFS}\), let \(\tau \in \{1, 2, \ldots, |V| + |C|\}\) denote the minimum integer such that \(v_\tau\) either is a sibling of \(v_{|V| + |C|}\) or is \(v_{|V| + |C|}\) itself. (So \(v_\tau, v_{\tau+1}, \ldots, v_{|V| + |C|}\) are siblings.) Define \(\mathcal{P} \doteq \{i \mid \tau \leq i \leq |V| + |C|, \omega(v_i) \leq k\}\) and \(\mathcal{Q} \doteq \{i \mid \tau \leq i \leq |V| + |C|, \delta(v_i) \leq k\}\). Since \(\forall v \in V\), either \(\delta(v)\) or \(\omega(v)\) is infinite but not both, \(\mathcal{P}\) and \(\mathcal{Q}\) form a partition of the set \(\{\tau, \tau + 1, \ldots, |V| + |C|\}\).

By convention, for the empty set \(\emptyset\), we say \(\max_{i \in \emptyset} \delta(v_i) = \max_{i \in \emptyset} \omega(v_i) = 0\). We first make some observations.

Lemma 35. Suppose \(\max_{i \in \mathcal{P}} \omega(v_i) > \max_{i \in \mathcal{Q}} \delta(v_i)\). Let \(i^*\) be an integer in \(\mathcal{P}\) such that \(\omega(v_{i^*}) = \max_{i \in \mathcal{P}} \omega(v_i)\). Then there exists a minimum-sized \(g\)-Elimination Set for \(G_{BFS}\) that includes the nodes in \(\{v_i \mid i \in \mathcal{P}, i \neq i^*\}\) but not \(v_{i^*}\).

Proof: Any \(g\)-Elimination Set for \(G_{BFS}\) has to include at least \(|P| - 1\) nodes in \(\{v_i \mid i \in \mathcal{P}\}\) because otherwise the un-included nodes in \(\{v_i \mid i \in \mathcal{P}\}\) cannot be corrected (using the check node \(\pi(v_{|V| + |C|})\)). If \(|T| = |P| - 1\), let \(j \in \mathcal{P}\) be an integer such that \(v_j \not\in T\). If \(T \cap \{v_i \mid i \in \mathcal{Q}\} = \emptyset\), then \(v_j\) cannot be corrected by iteration \(\omega(v_j) \leq \max_{i \in \mathcal{P}} \omega(v_i) \leq \max_{i \in \mathcal{Q}} \delta(v_i)\) because not all nodes in \(\{v_i \mid i \in \mathcal{Q}\}\) will be corrected by iteration \(\omega(v_j) - 1\), so this is an impossible case. So \(T \cap \{v_i \mid i \in \mathcal{Q}\} \neq \emptyset\). Let \(m \in \mathcal{Q}\) be an integer such that \(v_m \in T\); then we can replace \(v_m\) by \(v_j\) in \(T\) and get another \(g\)-Elimination Set \(T'\) for \(G_{BFS}\) because \(v_m\) helps decoding more than \(v_j\); \(v_m\) can be corrected automatically. Since \(|T'| = |T|\) and \(\{v_i \mid i \in \mathcal{P}\} \subseteq T'\), the conclusion holds.

The next two lemmas show how to reduce the \(gSSE\) Problem from \(G_{BFS}\) to its subtree \(G_{sub}\). In some cases, in the derived \(gSSE\) Problem for \(G_{sub}\), the values of \(\delta(\pi(\pi(v_{|V| + |C|})))\) and \(\omega(\pi(\pi(v_{|V| + |C|})))\) in \(G_{sub}\) may be different from their original values in \(G_{BFS}\); and in such cases, to avoid confusion, we will denote the tree \(G_{sub}\) by \(G_{sub}\).

Lemma 37. Suppose \(\max_{i \in \mathcal{P}} \omega(v_i) \leq \max_{i \in \mathcal{Q}} \delta(v_i)\). Consider five cases:

1) \(\mathcal{Q} = \emptyset\), \(\delta(v_{|V| + |C|}) = |P| - 1\), let \(S\) be a minimum-sized \(g\)-Elimination Set for \(G_{sub}\).
2) \(\max_{i \in \mathcal{P}} \omega(v_i) > \max_{i \in \mathcal{Q}} \delta(v_i) > \max_{i \in \mathcal{Q}} \delta(v_i)\).
3) \(\max_{i \in \mathcal{Q}} \delta(v_i) = |P| - 1\), let \(S\) be a minimum-sized \(g\)-Elimination Set for \(G_{sub}\).
4) \(\omega(\pi(\pi(v_{|V| + |C|}))) \leq \max_{i \in \mathcal{Q}} \delta(v_i)\).
5) \(\omega(\pi(\pi(v_{|V| + |C|}))) > \max_{i \in \mathcal{Q}} \delta(v_i)\).

Proof: By Lemma 36, there exists a minimum-sized \(g\)-Elimination Set for \(G_{BFS}\) that contains all the nodes in \(\{v_i \mid i \in \mathcal{P}\}\). Now consider only minimum-sized \(g\)-Elimination Sets for
$G_{BFS}$ that contain all the nodes in \{v_i| i \in P\}. See the nodes in \{v_i| i \in P\} as removed (because nodes in an Elimination Set are removed before decoding begins); then to prove the conclusion, we just need to prove this assertion: when P = 0, S is a minimum-sized g-Elimination Set for $G_{BFS}$.

For Case 1, since $\max_{i \in Q} \delta(v_i) = k$, the subtree rooted at $\pi(v_i|v_i+|C|)$ cannot help correct the node $\pi(v_i|v_i+|C|))$ in the first k iterations. Every node v with $\omega(v) \neq \infty$ is in $G_{sub}$ and has $\omega(v) \leq k$. So finding a minimum-sized g-Elimination Set for $G_{BFS}$ is equivalent to finding such as set for $G_{sub}$. So the assertion holds.

For Case 2, if we compare $G_{sub}$ and $\hat{G}_{sub}$, we see that they differ only in their values of $\delta(\pi(v_i|v_i+|C|))$. (For $G_{sub}$, that value is $\min(\delta(\pi(v_i|v_i+|C|)), \max_{i \in Q} \delta(v_i) + 1).$ Now observe the check node $\pi(v_i|v_i+|C|)$ and its neighboring variable nodes: when BP decoder runs on $G_{BFS}$, all the nodes in \{v_i| i \in Q\} can be corrected automatically by iteration $\max_{i \in Q} \delta(v_i) < k$; so by using the check node $\pi(v_i|v_i+|C|)$, the node $\pi(v_i|v_i+|C|)$ can be corrected by iteration $\max_{i \in Q} \delta(v_i) + 1 \leq k$. That is equivalent to turning $\delta(\pi(v_i|v_i+|C|))$ into $\min(\delta(\pi(v_i|v_i+|C|)), \max_{i \in Q} \delta(v_i) + 1)$ and turning $G_{sub}$ into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the assertion. The remaining three cases can be proved similarly. (For details, please see [3].)

Lemma 38. Suppose $\max_{i \in P} \omega(v_i) > \max_{i \in Q} \delta(v_i)$. Let $i^*$ be an integer in P such that $\omega(v_{i^*}) = \max_{i \in P} \omega(v_i)$. Consider two cases:

1) Case 1: If $\max_{i \in P} \omega(v_i) > \delta(\pi(v_i|v_i+|C|))$, let $S$ be any minimum-sized g-Elimination Set for $G_{sub}$.

2) Case 2: If $\max_{i \in P} \omega(v_i) \leq \delta(\pi(v_i|v_i+|C|))$, let $S$ be any minimum-sized g-Elimination Set for $\hat{G}_{sub}$ where $\delta(\pi(v_i|v_i+|C|))$ is changed to $\infty$ and $\omega(v_i)$ is changed to $\min(\omega(v_i), \omega(v_{i^*}))$. Then $S \cup \{v_i| i \in P, i \neq i^*\}$ is a minimum-sized g-Elimination Set for $G_{BFS}$.

Proof: By Lemma 35, there exists a minimum-sized g-Elimination Set for $G_{BFS}$ that includes the nodes in \{v_i| i \in P, i \neq i^*\} but not $v_{i^*}$. Let $T^*$ be such a minimum-sized g-Elimination Set for $G_{BFS}$.

For Case 1, when the g-Elimination Set for $G_{BFS}$ is $T^*$, the subtree rooted at the check node $\pi(v_i|v_i+|C|)$ cannot help correct the node $\pi(v_i|v_i+|C|))$. Instead, those nodes of $T^*$ that are in $G_{sub}$ will be a g-Elimination Set for $G_{sub}$, and the BP decoder will correct the un-removed nodes in $G_{sub}$ (within each of their required number of iterations $\omega(v_i)$). If $\pi(\pi(v_i|v_i+|C|))$ is in $T^*$, the check node $\pi(v_i|v_i+|C|)$ will correct $v_{i^*}$ in the 1st iteration; otherwise, the BP decoder will correct $\pi(\pi(v_i|v_i+|C|))$ in at most $\delta(\pi(v_i|v_i+|C|))$ iterations, so $\pi(v_i|v_i+|C|)$ will correct $v_{i^*}$ in at most $\delta(\pi(v_i|v_i+|C|)) + 1 \leq \omega(v_{i^*})$ iterations. Since $T^*$’s size is minimized, the number of nodes of $T^*$ that are in $G_{sub}$ is also minimized. That leads to the conclusion.

For Case 2, when the g-Elimination Set for $G_{BFS}$ is $T^*$, the BP decoder needs to correct the node $\pi(v_i|v_i+|C|))$ by iteration $\max_{i \in P} \omega(v_i) - 1 < \delta(\pi(v_i|v_i+|C|))$ because only then will the check node $\pi(v_i|v_i+|C|))$ help correct the node $v_{i^*}$ by iteration $\max_{i \in P} \omega(v_i) = \omega(v_{i^*})$. That is equivalent to turning $\omega(\pi(v_i|v_i+|C|))$ into $\min(\omega(\pi(v_i|v_i+|C|)), \max_{i \in P} \omega(v_i) - 1)$, turning $\delta(\pi(v_i|v_i+|C|))$ into $\infty$ and turning $G_{sub}$ into $\hat{G}_{sub}$, when it comes to BP decoding. That leads to the conclusion.

We can design an algorithm for $SS\infty_k$ as follows: (1) run BFS on G to get $G_{BFS}$ and as initialization, let $\omega(v) = k$ and $\delta(v) = \infty$ for every $v \in V$; (2) use Lemma 37 and 38 repeatedly to reduce the graph in the $gSS$ Problem from $G_{BFS}$ to its subtree $G_{sub}$ (or $\hat{G}_{sub}$), and then to smaller and smaller subtrees in the same way, until the subtree contains only the root node $v_1$; during this reduction process, more and more nodes (namely either $v_i| i \in P$) in Lemma 37 or $v_i| i \in P, i \neq i^*$ in Lemma 38) are included in the Elimination Set; (3) in the last step, when the subtree contains only $v_1$, include $v_1$ in the Elimination Set if and only if $\omega(v_1) \leq k$ at that moment. The above algorithm can be implemented by processing the nodes in the reverse order of their labels – from $v_i|v_i+|C|$ back to $v_1$ – and has time complexity $O(|V| + |C|)$. Due to space constraints, we omit its pseudo code here. We can see that it returns an optimal (i.e., minimum-sized) k-iteration Elimination Set of G = (V U C, E).

For the special case of $k = \infty$, the algorithm can be simplified: for every check node, include all but one of its children in the Elimination Set S; also include $v_1$ in S. For its details, please see [3].

VI. CONCLUSIONS

This paper studies the Stopping-Set Elimination Problem motivated by several applications. The NP-hardness of both $SS\infty_k$ and $SS\in k$ Problems is proven. An approximation algorithm is presented for the $SS\infty_k$ Problem. And linear-time algorithms that return optimal solutions are presented for the $SS\infty_k$ and $SS\in k$ Problems when the Stopping Sets have tree structures.

REFERENCES


