Stopping Set Elimination for LDPC Codes

Anxiao (Andrew) Jiang*, Pulakesh Upadhyaya*, Ying Wang⋆, Krishna 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 $k = 1$. And efficient exact algorithms are presented for general k when the stopping sets form trees.**

I. Introduction

The amount of data stored in the Internet is growing exponentially fast. With this growth, how to ensure long-term data reliability for all data also becomes more challenging. To assist error-correcting codes (ECC), the redundancy in the content of data itself can be utilized. This type of redundancy – such as features in languages, images and videos, structures in HTML files and databases, etc. – is referred to as *natural redundancy (NR)*, which supplements the more structured redundancy added by error-correcting codes [12], [13]. NR exists in both uncompressed and imperfectly compressed data, which are abundant in storage systems. That makes NR a promising tool to enhance data reliability.

With NR, a decoding system can be considered as consisting of two decoders: an ECC-Decoder, and an NR-Decoder. They work collaboratively to correct errors or erasures in the ECC codeword. We illustrate it by an example.

Example 1. *Consider texts compressed by an LZW algorithm that uses a fixed dictionary of size 2^ℓ. The dictionary has 2^ℓ text strings (called patterns) of variable lengths, where every pattern is encoded as an ℓ-bit codeword. Given a text to compress, the LZW algorithm scans T and partitions it into patterns, and maps them to codewords. For instance, if $\ell = 20$ and the text is "Flash memory is an electronic $\cdots$", the partitioning and LZW-codewords can be as illustrated in Fig.* 1 *(a).*

Now suppose some bits in the LZW-codewords are erased. An NR-Decoder can check all the possible solutions, map each solution back to patterns, and use a dictionary of words to eliminate those solutions that contain invalid words. (Such a dictionary of words has been commonly used in spell checkers.) If all the remaining solutions agree on the value of an erased bit, then that erasure is decoded by the NR-Decoder. For instance, suppose each LZW-codeword in Fig. 1 *(a) suffers from two erasures, which lead to four possible solutions/patterns (see Fig.* 1 *(b)). By combining the patterns for each codeword, we can rule out many solutions. For instance, the combination "should becnomially ars an ele" can be eliminated due to the invalid word "becnomially". In fact, the only combination without invalid words (without considering words on the boundary of the string, which might be part of a longer word) is "Flash memory is an ele", so the NR-Decoder can recover all six erasures in the three codewords.*

Suppose that the LZW-codewords, seen as information bits, are protected by a systematic ECC. Then the ECC-Decoder can correct erasures by parity-check constraints, and the NR-Decoder can correct erasures by NR. They can work collaboratively to maximize the number of correctable erasures. □

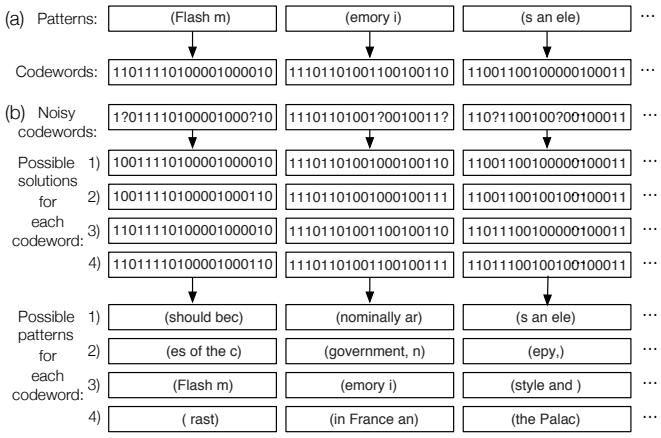

Fig. 1. (a) Compress a text by LZW. (b) NR-decoding for erasures.

As this paper is motivated by language-based NR, we would like to mention that an LZW algorithm with a dictionary of 2^{20} patterns (as in the above example) can compress the English language to 2.94 bits per character. The UNIX Compress command uses LZW with a smaller dictionary and so achieves a lower compression ratio. There are compression algorithms for languages with higher compression ratios (e.g., syllable-based Burrows-Wheeler Transform achieving 2 bits/character [15]). However, there is still a gap toward Shannon's estimation of 1.34 bits/character for the entropy of English [28], which gives motivation for NR-Decoders. And one may reasonably conjecture that a similar scenario exists for images and videos.

In this work, we propose a relatively generic decoding

model for collaborative ECC-Decoding and NR-Decoding that is motivated by language-based NR. The model is shown in Fig. 2. The (compressed or uncompressed) data, seen as information bits, are encoded into a systematic ECC codeword. The NR-decoder uses a sliding window of L bits to check a segment of the data each time, and uses its NR to correct errors/erasures in it. We bound the size of the window to L bits because due to the lack of structures in NR, NR-decoding is often not as efficient as ECC-decoding and its complexity grows with L, so a finite L bounds the acceptable complexity of NR-decoding. The NR-Decoder works jointly with the ECC-Decoder to correct errors/erasures.

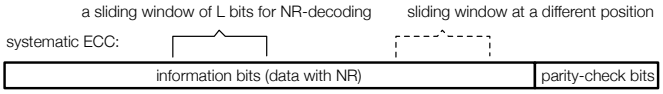

Fig. 2. A model for collaborative ECC-decoding and NR-decoding.

The above model can be applied to languages compressed by LZW codes or Huffman codes, where some practical decoding algorithms have been presented [12], [13], [17], [19], [31], [32]. In this paper, we study a basic theoretical problem for LDPC codes: when the number of erasures in a noisy LDPC codeword exceeds the decoding capability of the LDPC code's ECC-Decoder, what is the minimum number of erasures that an NR-Decoder needs to help correct so that the remaining erasures are decodable by the ECC-Decoder?

Let us define the problem more specifically. Let the LDPC code's ECC-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 ECC-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. 3. 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. 3 (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. 3 (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.

We now define the capability and limitations of the NR-Decoder. Suppose that for any sliding window of L bits, the NR-Decoder can always correct its erasures if the number of

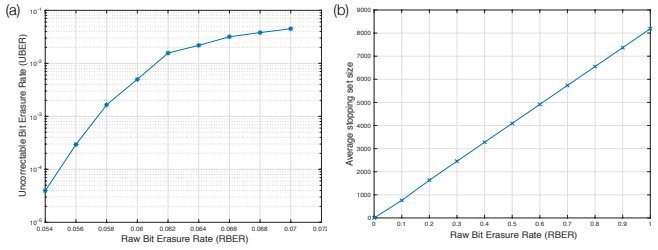

Fig. 3. Statistics of an (8192,7561) LDPC code. (a) UBER for different RBERs near the code's decoding threshold. (b) Average stopping-set size for different RBERs.

erasures in the window is at most α. Given a stopping set, the objective is to use the NR-Decoder to correct sufficiently many erasures so that the remaining erasures are correctable by the ECC-Decoder. However, notice that since NR-Decoding is typically less efficient than ECC-Decoding, there is an associated cost. Let β denote the number of L-bit windows (at β different locations) used by NR-Decoding. Whether the NR-decoding is implemented in hardware or software (where decoding circuits or software can choose the location of each window), the overall circuit complexity and/or time complexity is proportional to β. (For instance, if circuits are used to decode the windows in parallel, then β circuits are needed for β windows.) Therefore, we need to *minimize* the number of windows used by NR-decoding, and choose the locations of the windows carefully for that purpose.

In this paper, we will study a special case of the above problem by setting $L = \alpha = 1$, and we assume that the sliding windows can cover both information bits and parity-check bits. Although it may seem too restrictive at first sight, there are several reasons that still make it quite meaningful. First, storage systems often have low raw bit-error-rates (e.g., less than 0.5%), so for a relatively short window (e.g, tens of bits), the number of erasures in it is often at most 1, which can usually be corrected very effectively by NR-Decoding [13]. In such cases, having $L \geq 1$, $\alpha = 1$ is similar to having $L = \alpha = 1$. Second, ECCs in storage systems often have high rates (e.g., over 0.93), which can make the sliding window's access to all codeword bits similar to accessing only information bits (since information bits are the majority of bits). Third, understanding the basic case of $L = \alpha = 1$ will be the basis for understanding the more general case of $L \geq \alpha \geq 1$. And last but not least, the case $L = \alpha = 1$ corresponds to a fundamental problem for LDPC codes: assume there is a powerful and unrestricted Oracle decoder that can correct any erasure, but its decoding comes at a high cost; then, how to minimize the number of erasures the Oracle decoder needs to correct in order to make the remaining erasures decodable by the ECC-Decoder? We believe the problem is theoretically important in its own right.

The problem to study can now be defined formally as follows. Let $G = (V \cup C, E)$ be a bipartite graph, where V (representing erasures) is a subset of the variable nodes in an LDPC code's Tanner graph, C is a subset of the check

nodes in the same Tanner graph such that every node in C is adjacent to at least one node in V, and E is the set of edges in the Tanner graph with one endpoint in V and another 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*. 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), then V is called a *Decodable Set* (or simply *decodable*); otherwise, it is a *Non-Decodable Set* (or simply *non-decodable*). Note that a Stopping Set must be a Non-Decodable Set, but not *vice versa*. The problem we study, called *Stopping-Set Elimination (SSE) Problem*, is as follows.

Definition 2. *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 are decodable?*

The removed variable nodes represent NR-decoded erasures. Clearly, after the removal, the remaining nodes will no longer contain any Stopping Set.

The rest of the paper is organized as follows. In Section II, we introduce more applications of the SSE Problem, and also review related works. In Section III, we prove the NR-hardness of the SSE Problem. In Section IV, we study a variation of the SSE Problem, where the remaining variable nodes are required to be decodable by BP decoding within a constant number of iterations, and prove its NP-hardness. In Section V, we present an approximation algorithm for the latter problem. In Section VI, we present practical algorithms for the SSE Problem. In Section VII, we present concluding remarks.

II. Related Applications and Related Works

In this section, we first show two additional applications of the SSE Problem. We then present a brief review of existing works that are related to error correction by NR.

A. Applications of SSE

In addition to decoding by NR, the SSE Problem also has two additional applications: distributed storage, and satellite-to-ground communication with feedback.

1) Distributed Storage: Distributed file systems like HDFS have been widely used in big data applications [29]. 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. And that is the SSE Problem.

2) Satellite-to-Ground Communication with Feedback: Consider satellite-to-ground communication, where the 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) [20]. As the channel is noisy, some packets received by the ground may be undecodable, and the ground will request the satellite to retransmit some of those lost packets. Since the satellite-to-ground communication can be quite costly, it is desirable to minimize the number of retransmitted packets. That is also the SSE Problem.

B. Related Works

Error-correction with NR is related to joint source-channel coding and denoising. The idea of using the inherent redundancy in a source – or the leftover redundancy at the output of a source encoder – to enhance the performance of the ECC has been studied within the field of joint source-channel coding. In [10], source-controlled channel coding using a soft-output Viterbi algorithm is considered. In [3], a trellis based decoder is used as a source decoder in an iterative decoding scheme. Joint decoding of Huffman and Turbo codes is proposed in [9]. In [11], joint decoding of variable length codes (VLCs) and convolutional/Turbo codes is analyzed. Applications of turbo codes to image/video transmission are shown in [7], [23] and [14]. Joint decoding using LDPC codes for VLCs and images are illustrated in [24] and [25], respectively. However, not many works have considered JSCC specifically for language-based sources, and exploiting the redundancy in the language structure via an efficient decoding algorithm remains as a significant challenge. Related to joint source-channel coding, denoising is also an interesting and well studied technique [2], [4], [5], [6], [18], [21], [22], [26], [34]. A denoiser can use the statistics and features of input data to reduce its noise level for further processing. For discrete memoryless channels with stationary input sequences, a universal algorithm that performs asymptotically as well as optimal denoisers are given in [33]. The algorithm is also universal for a semi-stochastic setting, where the channel input is an individual sequence and the randomness in the channel output is solely due to the channel's noise.

Spell-checking softwares are a typical example of using NR to correct errors in languages. They are widely used in text editors. A spell-checking software usually works at the character level (namely, it does not consider how characters or text strings are encoded by bits), is for uncompressed texts, and uses the validity of words and the correctness of grammar to correct errors that appear in the typing of texts.

Using NR to correct errors at the bit level in compressed texts has been studied in a number of works. In [17], texts compressed by Huffman coding is considered, and a dynamic programming algorithm is used to partition the noisy bit sequence into subsequences that represents words, and to

select likely solutions based on the frequencies of words and phrases. In [12], texts that are compressed by Huffman coding and then protected by LDPC codes are studied. An efficient greedy algorithm is used to decompress the noisy bit string, and partition it into stable and unstable regions based on whether each region contains recognizable words and phrases. The stable and unstable regions have polarized RBERs, which are provided as soft information to the LDPC code for better decoding performance. The algorithm is enhanced in [19] by a machine learning method for content recognition, and an iterative decoding algorithm between the NR-Decoder and the ECC-Decoder is used to further improve performance. In [32], texts compressed by Huffman coding and protected by Polar codes are studied. The validity of words is used to prune branches in a list sequential decoding algorithm, and a trie data structure for words is used to make the algorithm more efficient. A concatenated-code model that views the text with NR as the outer code and the Polar code as the inner code is considered, and the rate improvement for the Polar code due to NR is analyzed. That model is further studied in [31], where an optimal algorithm that maximizes the code rate improvement by unfreezing some frozen bits to store information is presented. A model that views NR as the output of a side information channel at the channel decoder is also studied, where NR is shown to improve the random error exponent.

III. NP-Hardness of SSE Problem

In this section, we prove that the SSE 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 Problem.

A. NP-completeness of Pseudo Set Cover Problem

Consider the well-known Set Cover Problem. Let $T = \{t_1, t_2, \cdots, t_n\}$ be a universe of n elements. Let $S_1, S_2, \cdots, S_m$ be m subsets of T such that $T = \bigcup_{i=1}^{m} S_i$. Let $k \leq m$ be a positive integer. The Set Cover Problem asks if there exist k subsets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ such that $T = \bigcup_{j=1}^{k} S_{i_j}$. (Note that a subset S_i is said to "cover" its elements. So the Set Cover Problem asks if there exist k subsets that together cover all the elements of T.)

Let us now define a related problem called the *Pseudo Set Cover Problem*. It has the same input as the Set Cover Problem, and differs only in its question: it asks if there exist k subsets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ such that $|\bigcup_{j=1}^{k} S_{i_j}| \geq |T| - 1$. (Instead of covering all the $|T|$ elements, the Pseudo Set Cover Problem aims at covering at least $|T| - 1$ elements.) We now prove that the problem is NP-complete.

Theorem 3. *The Pseudo Set Cover Problem is NP-complete.*

Proof: It is easy to see that the Pseudo Set Cover Problem is in NP. We now construct a polynomial-time reduction from the Set Cover Problem to the Pseudo Set Cover Problem.

Let an instance of the Set Cover Problem have input parameters $T = \{t_1, t_2, \cdots, t_n\}$, S_1, S_2, $\cdots$, 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

$$T' = \{t_1, t_2, \cdots, t_n, t_{n+1}\},$$

where t_{n+1} is a new element, and let its subsets be

$$S_1, S_2, \cdots, S_m, S_{m+1},$$

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

$$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$$

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 $\{S_1, S_2, \cdots, S_m\}$, 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 $S_{i_k} = S_{m+1}$ 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.

Therefore there is a polynomial-time reduction from the Set Cover Problem to the Pseudo Set Cover Problem. Since the Set Cover Problem is known to be NP-complete, the conclusion holds. ∎

B. NP-hardness of Stopping Set Elimination Problem

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

Consider the bipartite graph shown in Fig. 4 (a). It consists of four variable nodes (s_i, t_j, $u_{i,j}$ and $w_{i,j}$) and three check nodes ($c^1_{i,j}$, $c^2_{i,j}$ and $c^3_{i,j}$). 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.

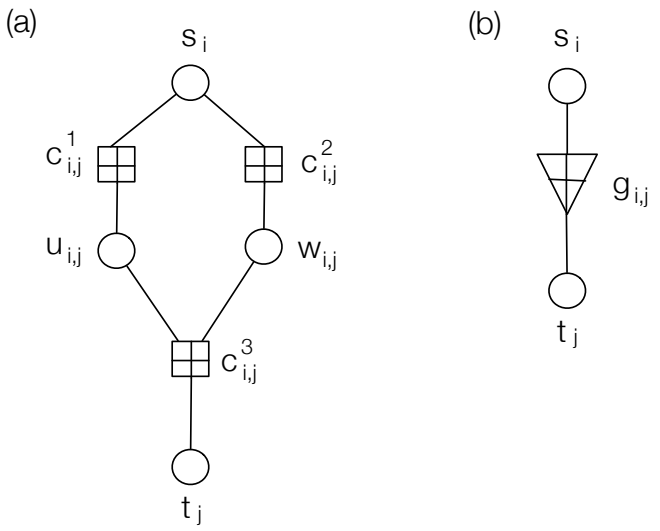

Fig. 4. (a) A bipartite graph $D_{i,j}$ that connects variable nodes s_i and t_j. (b) A symbol for the graph $D_{i,j}$.

Lemma 4. *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.*

On the other hand, if the value of the variable node t_j becomes known, the BP decoding algorithm will not recover the value of any of the other three variable nodes.

Proof: If the value of s_i becomes known, by using the check nodes $c^1_{i,j}$ and $c^2_{i,j}$, the BP decoding algorithm will recover the values of $u_{i,j}$ and $w_{i,j}$, respectively. Then via the check node $c^3_{i,j}$, it will recover the value of t_j.

If the value of t_j becomes known, since $c^3_{i,j}$ has degree 3, the BP algorithm will not recover any more values. ■

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. 4 (b), where the "gate" $g_{i,j}$ represents the five nodes ($c^1_{i,j}$, $c^2_{i,j}$, $c^3_{i,j}$, $u_{i,j}$, $w_{i,j}$) 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 $T = \{t_1, t_2, \cdots, t_n\}$, S_1, S_2, $\cdots$, S_m and $k \leq m$ as introduced earlier. To reduce it to the SSE Problem, we will map every instance of the Pseudo Set Cover Problem to some instance of the SSE Problem.

Let us start by building a graph G_I. 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_I. 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_I. An example is shown below.

Example 5. *Let an instance of the Pseudo Set Cover Problem be $T = \{t_1, t_2, t_3, t_4, t_5\}$ and $S_1 = \{t_1, t_3, t_4\}$, $S_2 = \{t_1, t_3\}$, $S_3 = \{t_2, t_4, t_5\}$. The value of parameter k is irrelevant to the mapping, so we do not specify it here. The instance is illustrated in Fig. 5 (a), where there is an edge between S_i and t_j if and only if $t_j \in S_i$.*

The corresponding graph G_I is shown in both Fig. 5 (b) and (c), where the symbol for each $D_{i,j}$ is used in Fig. 5 (b), and the full details of G_I are shown in Fig. 5 (c). It is easy to see the correspondence between G_I and the Pseudo Set Cover Problem. □

It is clear that G_I is a bipartite graph.

Let us now create a graph G_{II} as follows. Given graph G_I, we add $m+1$ additional check nodes

$$c_0, c_1, c_2, \cdots, c_m.$$

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_{II}. An example is shown below.

Example 6. *Following Example 5, the graph G_{II} is shown in Fig. 5 (d).* □

It is clear that G_{II} is also a bipartite graph.

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_{II}, the degree of every check node is at least two. So it is a Stopping Graph, namely, an instance of the SSE Problem.

Lemma 7. *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_{II}, k variable nodes can be removed so that the remaining variable nodes form a Decodable Set.*

Proof: Suppose that

$$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$$

are k chosen subsets that cover at least $n-1$ elements of T. Let us remove the corresponding k variable nodes

$$s_{i_1}, s_{i_2}, \cdots, s_{i_k}$$

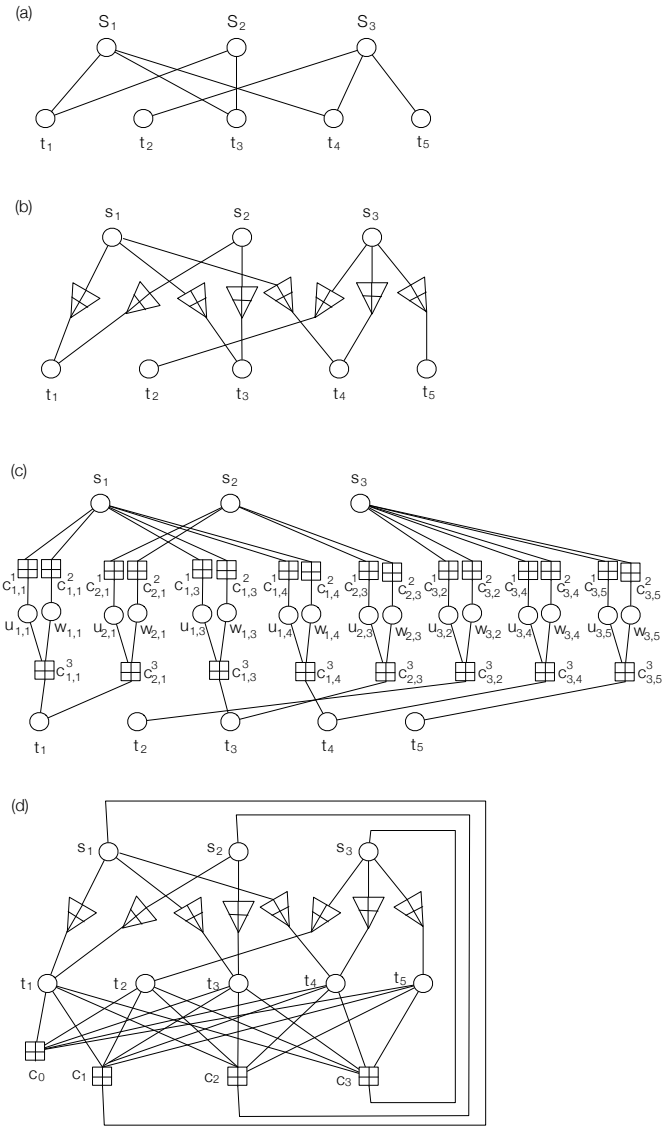

Fig. 5. (a) An instance of the Pseudo Set Cover Problem, where $T = \{t_1, t_2, t_3, t_4, t_5\}$ and $S_1 = \{t_1, t_3, t_4\}$, $S_2 = \{t_1, t_3\}$, $S_3 = \{t_2, t_4, t_5\}$. (b) The corresponding graph G_I. (c) The corresponding graph G_I with full details. (d) The corresponding graph G_{II}.

from the graph G_{II}. 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

$$t_1, t_2, \cdots, t_n.$$

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 $t_1, t_2, \cdots, t_n$. 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 $t_1, t_2, \cdots, t_n$ are all known now, for $i = 1, 2, \cdots, m$, 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 a Decodable Set, let us call S an *Elimination Set* of size $|S|$.

Lemma 8. *If G_{II} has an Elimination Set of size $k \leq m$, then G_{II} has an Elimination Set of size k that is also a subset of*

$$\{s_1, s_2, \cdots, s_m\}.$$

.

Proof: Let

$$X = \{x_1, x_2, \cdots, x_k\}$$

be an Elimination Set of G_{II}, where each x_i is a variable node. Let us create a set

$$Y = \{y_1, y_2, \cdots, y_k\} \subseteq \{s_1, \cdots, s_m\}$$

as follows. For $i = 1, 2, \cdots, k$, do:

- If $x_i \in \{s_1, s_2, \cdots, s_m\}$, let $y_i = x_i$.
- If x_i is either $u_{i',j'}$ or $w_{i',j'}$ – namely, it is a variable node in the "gadget" $D_{i',j'}$ (more specifically, $g_{i',j'}$) that connects $s_{i'}$ and $t_{j'}$ – let $y_i = s_{i'}$ if $s_{i'}$ is not in Y yet, and let y_i be any node in $\{s_1, s_2, \cdots, s_m\}$ that is not yet in Y otherwise.
- If $x_i = t_j$ for some $1 \leq j \leq n$, let $s_{i'}$ be a node such that there is a "gadget" $D_{i',j}$ connecting $s_{i'}$ and t_j. (Such a node $s_{i'}$ must exist because in the Pseudo Set Cover Problem, t_j is covered by at least one subset.) If $s_{i'}$ is not in Y yet, let $y_i = s_{i'}$; otherwise, let y_i be any node in $\{s_1, s_2, \cdots, s_m\}$ that is not yet in Y.

With the above construction, for any node x_i in X, there exists a node $s_{i'}$ in Y such that either $s_{i'} = x_i$, or $s_{i'}$ and x_i exist in the same "gadget" $D_{i',j}$ for some j. By the "directed" property of gadgets $D_{i',j}$, we see that when the values of variable nodes in Y are known, the BP algorithm can decode all the variable nodes in X; and since X is an Elimination Set, the BP algorithm can consequently decode all the variable nodes in G_{II}. So Y is an Elimination Set of size k that is a subset of $\{s_1, s_2, \cdots, s_m\}$. ■

Lemma 9. *If G_{II} has an Elimination Set of size k $\{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\} \subseteq \{s_1, s_2, \cdots, s_m\}$, then for the corresponding Pseudo Set Cover Problem, the k subsets*

$$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$$

cover at least $n-1$ elements of T.

Proof: The proof is by contradiction. Suppose that $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ cover at most $n-2$ elements of T. Then in G_{II}, when the values of $\{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$ are known, the BP algorithm can use the "gadgets" $D_{i,j}$ to decode at most $n-2$ variable nodes among

$$t_1, t_2, \cdots, t_n.$$

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 $\{t_1, t_2, \cdots, t_n\}$ 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 $s_i \notin \{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$, 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 $\{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$ is not an Elimination Set, which is a contradiction. That leads to the conclusion. ■

By combining the above two lemmas, we get:

Lemma 10. *If G_{II} 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 11. *The SSE Problem is NP-hard.*

Proof: The SSE 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 7 and Lemma 10, 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_{II} has an Elimination Set of size k). So the mapping is a polynomial-time reduction. By Theorem 3, the Pseudo Set Cover Problem is NP-complete. So P_{sse} is NP-complete, which leads to the conclusion. ■

IV. SSE with Constraint on Belief-Propagation Iterations and Its NP-Hardness

In this section, we extend the SSE Problem by considering the time for BP decoding. After the nodes in an Elimination Set are removed (namely, after NR-decoding corrects those erasures), the remaining erasures are guaranteed to form a Decodable Set, and therefore the BP decoder can correct them. However, there is no guarantee on *how many iterations* are needed by the BP decoder to correct the remaining erasures. Here we assume a standard parallel-implementation of BP decoding: in each iteration, first, all variable nodes transmit their values to neighboring check nodes in parallel; then, all check nodes use incoming messages to correct erasures and send the decoding results back to variable nodes, also in parallel. So the time for BP decoding can be measured by the number of BP iterations.

It can be seen that for a Stopping Set of n variable nodes (namely, n erasures), after an Elimination Set is removed, the BP decoder may still use as many as $\Theta(n)$ iterations to correct the remaining erasures. The example below is an illustration.

Example 12. *A stopping set of n variable nodes and n check nodes are shown in Fig. 6 (a), where all nodes have degree two and they together form a cycle. By eliminating one variable node v_1, the remaining variable nodes become decodable, as shown in Fig. 6 (b). Then the BP decoder corrects two variable nodes in each iteration: in the 1st iteration, it corrects v_2 and v_n because they both have a neighboring check node of degree one (as shown in Fig. 6 (c)); in the 2nd iteration, it corrects v_3 and v_{n-1} for the same reason (as shown in Fig. 6 (d)); and so on. So the BP decoder will use $\lceil \frac{n-1}{2} \rceil = \Theta(n)$ iterations to correct the remaining erasures.* □

For BP decoding, its decoding time is an important measure of performance. So it is useful to limit the number of iterations needed by BP decoding, which offers a performance guarantee. That motivates us to study this extended SSE Problem.

Definition 13. *Given a Stopping Graph $G = (V \cup C, E)$ and an integer k, how to remove the minimum number of variable nodes from V such that the remaining variable nodes can be corrected by the BP decoder in no more than k iterations?*

We call the above problem the SSE_k Problem. In comparison, the SSE Problem studied earlier has no constraint on k, so it can be seen as the SSE_∞ Problem.

We have already proved that SSE_∞ is NP-hard. The question now is: if k is a constant – namely, we want the BP decoding to finish within a fixed number of iterations – does the SSE_k problem become polynomial-time solvable? A positive answer seems possible at first sight, because having a small k puts more constraints on solutions and limits its search space. 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* (also called the *maximum subgraph problem*) [1], [8], [16], [30], which can be generally stated as follows: find the minimum number of vertices to delete from a given graph

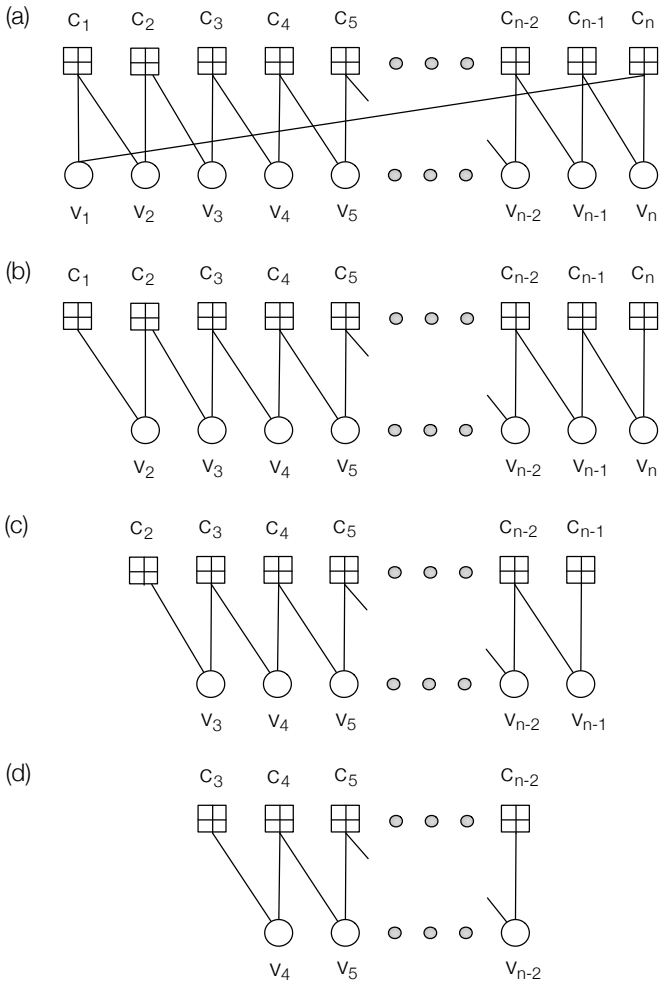

Fig. 6. (a) A Stopping Set of n variable nodes and n check nodes. (b) After removing a variable node v_1, the remaining nodes become decodable. (c) After the 1st iteration of BP decoding, v_2 and v_n are corrected. (d) After the 2nd iteration of BP decoding, v_3 and v_{n-1} are corrected.

so that the remaining subgraph satisfies a property π. The node-deletion problem includes many well-known problems as special cases. Some examples are:

- Max Clique Problem: the property π is that the remaining subgraph is a complete graph.
- Feedback Vertex Set Problem: the property π is that the remaining subgraph has no cycles.
- Vertex Cover Problem: the property π is that the remaining subgraph contains only isolated nodes, without edges.

Some node-deletion problems are NP-complete on both general graphs and bipartite graphs, such as the feedback vertex set problem. However, some are NP-complete on general graphs but polynomial-time solvable on bipartite graphs, such as the vertex cover problem.

The SSE_k Problem is different from the previously studied problems in several ways. First, its property π is for the remaining subgraph to be decodable within k iterations, which is different from the property π in other problems. Second, the previous works focus on properties π that are *hereditary on induced subgraphs*, namely, whenever a graph G satisfies π, by deleting nodes from G, the remaining subgraphs also satisfies π [1], [8], [16], [30]. (For example, the property π for the max clique problem is hereditary because when nodes are removed from a complete graph, the remaining subgraph is also a complete graph. The same holds for the feedback vertex set problem and the vertex cover problem.) However, for the SSE_k Problem, the property π is *not hereditary*, because when a check node is removed, it may turn a Decodable Set into a Non-decodable Set. An example is shown below.

Example 14. *A Decodable Set is shown in Fig. 7 (a), which satisfies the property π of the SSE_k problem. As shown in Fig. 7 (b), after the check nodes c_1 and c_3 are removed, the remaining subgraph becomes non-decodable, which violates the property π. So for the SSE_k Problem, the property π is not hereditary.* □

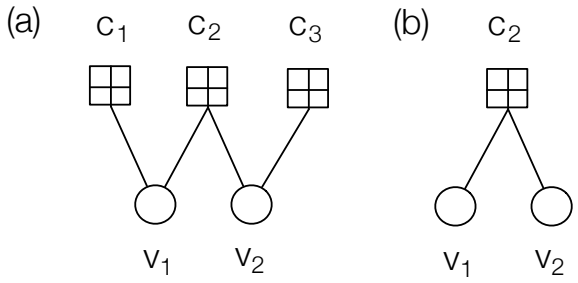

Fig. 7. (a) A graph with a Decodable Set. (b) After check nodes c_1 and c_3 are removed, the remaining variable nodes form a Non-decodable Set.

We now prove the NP-hardness of the SSE_1 Problem. We use a reduction from the NP-complete Not-all-equal SAT Problem [27], similar to a proof technique used in [30]. However, due to the differences between the SSE_k problem and the previously studied node-deletion problems (as mentioned above), the two proofs also have significant differences: they use different mappings from the Not-all-equal SAT Problem to the target problem, which also lead to some substantially different properties in the mapped structures.

We first define the Not-all-equal SAT Problem [27]: Let $x_1, x_2, \cdots, x_n$ be n Boolean variables. A *literal* is either x_i or $\bar{x}_i$ (namely, the NOT of x_i) for some $i \in \{1, 2, \cdots, n\}$. Let a *clause* be a set of three literals. Let

$$S = \{C_1, C_2, \cdots, C_k\}$$

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 15. *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* $C_1 = (x_1, \bar{x}_2, x_3)$*,* $C_2 = (\bar{x}_1, \bar{x}_2, x_4)$*,* $C_3 = (x_2, x_3, x_4)$*,* $C_4 = (x_1, \bar{x}_3, \bar{x}_4)$*,* $C_5 = (\bar{x}_1, x_2, x_3)$*.*

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).* □

We now construct a mapping from the Not-all-equal SAT Problem to the SSE_1 Problem. And afterwards, we will analyze its properties.

A. Reducing Not-all-equal SAT Problem to SSE_1 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_1 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. 8 (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_i^1 and X_i^0 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. 8 (b), which will be called gadgets U_j^1 and U_j^2, respectively. (Here for $t = 1, 2, 3$, nodes A_j^t and B_j^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. 8 (c), where for $t = 1, 2, 3$, two paths are used to connect the nodes A_j^t and B_j^t. (For example, the two paths between A_j^1 and B_j^1 have nodes d_j^1, d_j^2 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_i^1 and X_i^2. So to simplify the presentation, we sometimes represent V_i by the symbol in Fig. 8 (d), where the two "interface nodes" X_i^1, X_i^2 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_j^1, A_j^2, A_j^3, B_j^1, B_j^2, B_j^3; so we sometimes represent it by the symbol in Fig. 8 (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_j^t to X_i^1 and connect B_j^t to X_i^0 (through some intermediate nodes) as shown in Fig. 8 (f). Otherwise $l_t = \bar{x}_i$ for some

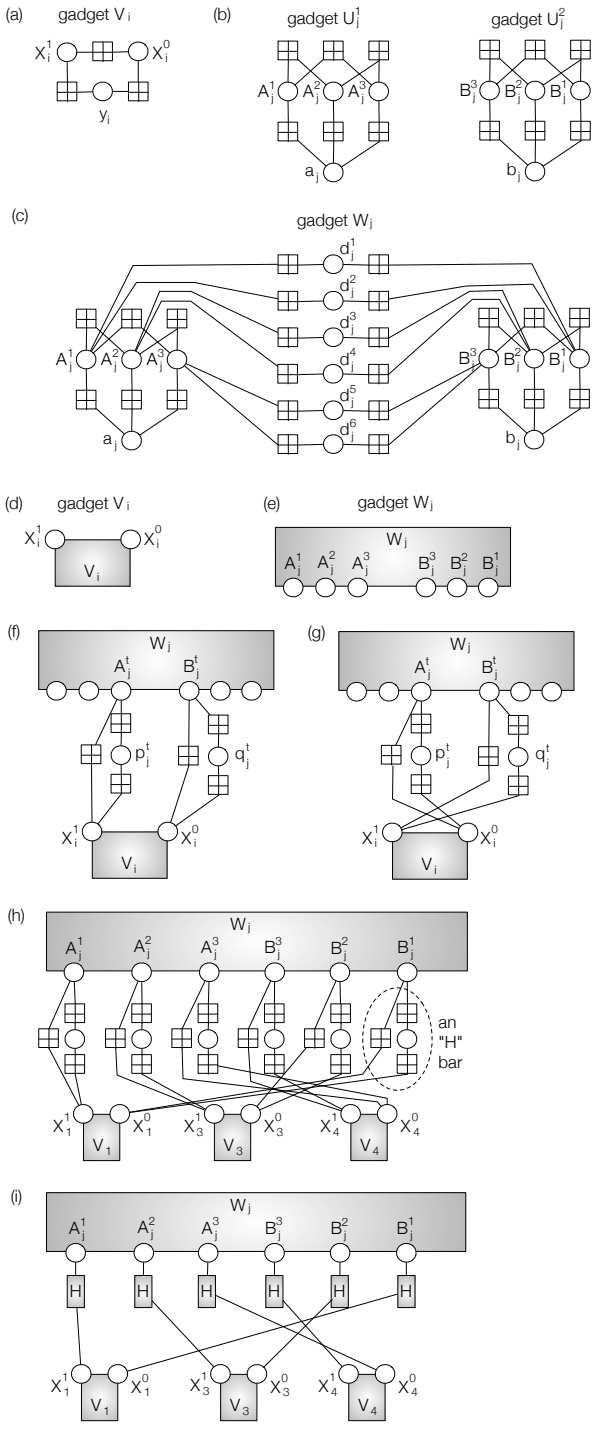

Fig. 8. (a) The gadget corresponding to a Boolean variable x_i, for $i = 1, 2, \cdots, n$. (b) Two gadgets corresponding to a clause C_j, for $j = 1, 2, \cdots, k$. (c) The connected gadget corresponding to a clause C_j, for $j = 1, 2, \cdots, k$. (d) Symbol for V_i. (e) Symbol for W_j. (f) Connect clause 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).

$1 \le i \le n$, and we connect A_j^t to X_i^0 and connect B_j^t to X_i^1 as shown in Fig. 8 (g).

Example 16. *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. 8 (h).*

To simplify the presentation of the graph, we represent the connection between a node A_j^t (or B_j^t) and a node x_i^1 (or x_i^0) by a rectangle that is generally denoted by the "H bar". Then the graph in Fig. 8 (h) is simplified as the presentation in Fig. 8 (i), which shows the connections more clearly. However, it should be noted that each A_j^t, B_j^t, x_i^1 or x_i^0 is connected to an H bar via two edges, not one. □

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.

Example 17. *For the Not-all-equal Problem, let $n = 4$ and $k = 2$. Let the two clauses be*

$$C_1 = (x_1, x_3, \bar{x}_4),$$
$$C_2 = (x_1, \bar{x}_2, \bar{x}_3).$$

Then the corresponding graph G_{sse} is shown in Fig. 9 (a), where its gadgets are represented by symbols for clarity, and also in Fig. 9 (b), where its full details are presented. □

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

In the previous subsection, the mapping from any instance of the Not-all-equal SAT Problem to a graph G_{sse} is shown. We now analyze its properties.

We first show a general property for the SSE_1 Problem.

Lemma 18. *Given a bipartite graph G (including the graph G_{sse}), where variable nodes represent erasures, the BP algorithm can decode all erasures in just one iteration if and only if for every variable node in G, it has at least one neighboring check node whose degree is 1.*

Proof: In each iteration of BP decoding, a check node c can recover the value of a neighboring variable node v if and only if v is its only neighboring erasure. ■

Note that it is unnecessary for all neighboring check nodes to have degree 1. For example, the graph in Fig. 7 can be decoded in one iteration, although both v_1 and v_2 have a neighboring check node c_2 that has degree two. In one iteration of BP decoding, the check node c_1 (which has degree 1) is sufficient for decoding v_1, and the check node c_3 (which also has degree 1) is sufficient for decoding v_2.

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 19. *Let*

$$\begin{aligned} I_{sse} \triangleq \quad & \{X_i^j \mid 1 \le i \le n, 0 \le j \le 1\} \cup \\ & \{A_i^j \mid 1 \le i \le k, 1 \le j \le 3\} \cup \\ & \{B_i^j \mid 1 \le i \le k, 1 \le j \le 3\} \end{aligned}$$

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. 9 (a).

Definition 20. *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 21. *If G_{sse} has a One-Iteration Elimination Set of α nodes, then G_{sse} also has a Canonical Elimination Set of at most α nodes.*

Proof: Let $F \subseteq V_{sse}$ be a One-Iteration Elimination Set of α nodes. We will prove the existence of a Canonical Elimination Set $\hat{F} \subseteq I_{sse}$ with $|\hat{F}| \le \alpha$ 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. 9 (a) for an illustration.) The main idea of the proof is to transform F into $\hat{F}$ by switching nodes of F to interface nodes.

First, consider a gadget V_i (for $1 \le i \le n$). (See Fig. 8 (a) for an illustration.) Note that X_i^1 and X_i^0 are its only two nodes connecting to nodes outside V_i in G_{sse}. (That is why X_i^1 and X_i^0 are called Interface Nodes.) If $y_i \in F$ (note that y_i is the other variable node in the gadget V_i), then consider three cases:

1) $X_i^1 \in F$ and $X_i^0 \in F$. In this case, 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}.
2) $X_i^1 \in F$ or $X_i^0 \in F$. Without loss of generality (WLOG), say $X_i^1 \in F$ and $X_i^0 \notin F$. In this case,

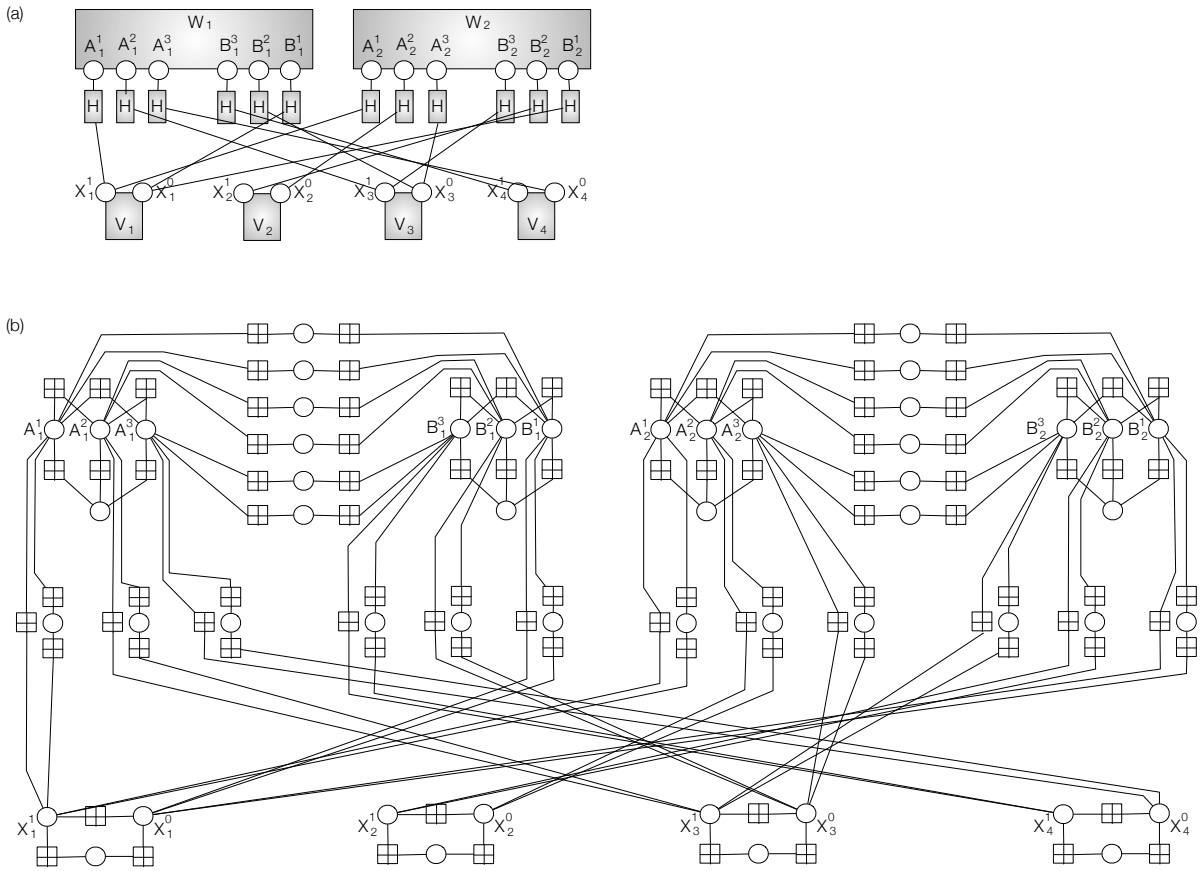

Fig. 9. (a) The graph G_{sse} corresponding to the Not-all-equal Problem where $n = 4$, $k = 2$, $C_1 = (x_1, x_3, \bar{x}_4)$, $C_2 = (x_1, \bar{x}_2, \bar{x}_3)$, where gadgets are represented by symbols. (b) The same graph G_{sse} in full detail.

we can replace y_i by X_i^0 in F and still get a one-iteration elimination set, because the replacement will only remove more edges from the part of the graph G_{sse} that is outside V_i, and y_i's two neighboring check nodes will have degree 1 after X_i^1 and X_i^0 are removed from G_{sse}.

3) $X_i^1 \notin F$ and $X_i^0 \notin F$. In this case, we can replace y_i by X_i^1 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 of degree 1, and more edges will be removed the part of the graph G_{sse} that is outside V_i.

So we can obtain a new one-iteration elimination set F_1 of at most α nodes, where

$$F_1 \cap \{y_i \mid 1 \le i \le n\} = \emptyset.$$

Next, consider an H bar. WLOG, suppose the H bar is between the two nodes A_j^t and X_i^1. (Such an H bar is illustrated in Fig. 8 (f).) Note that when the H bar is combined with the nodes A_j^t and X_i^1 (and the edges between them), we get a cycle that has the same structure as the gadget V_i (which is discussed above). So by the same argument that has been used for V_i, from F_1, we can obtain a new one-iteration elimination set F_2 of at most α nodes, where

$$F_2 \cap \{y_i \mid 1 \le i \le n\} = \emptyset,$$

$$F_2 \cap \{p_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset,$$

and

$$F_2 \cap \{q_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset.$$

Now consider a gadget W_j (for $1 \le j \le k$), which is shown in Fig. 8 (c). First, consider the two nodes d_j^1 and d_j^2, which are on two paths connecting A_j^1 and B_j^1. Those two paths together form a cycle of 4 variable nodes and 4 check nodes, with A_j^1 and B_j^1 as "interface nodes" connecting to the rest of the graph. (This cycle is quite similar to the cycle in gadget V_i.) With a similar analysis as the one for V_i, we can always turn F_2 into a new one-iteration elimination set by replacing d_j^1 and/or d_j^2 by A_j^1 and/or B_j^1. (The only slightly different case to note is when $A_j^1 \notin F_2$ and $B_j^1 \notin F_2$. In this case, we have $d_j^1 \in F_2$ and $d_j^2 \in F_2$, and we can replace them by A_j^1 and B_j^1 in F_2.) The same analysis applies to d_j^3, d_j^4, d_j^5 and d_j^6. So from F_2, we can obtain a new one-iteration elimination set F_3 of at most α nodes, where

$$F_3 \cap \{y_i \mid 1 \le i \le n\} = \emptyset,$$

$$F_3 \cap \{p_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset,$$
$$F_3 \cap \{q_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset$$

and

$$F_3 \cap \{d_j^t \mid 1 \le j \le k,\ 1 \le t \le 6\} = \emptyset.$$

Now consider the two component gadgets in W_j: U_j^1 and U_j^2, which are shown in Fig. 8 (b). WLOG, we just need to consider U_j^1. Suppose $a_j \in F_3$, and consider two cases:

1) $A_j^1 \in F_3$, $A_j^2 \in F_3$, or $A_j^3 \in F_3$. WLOG, suppose $A_j^1 \in F_3$. In this case, we can delete a_j from F_3 and still get a one-iteration elimination set, because after A_j^1 is removed from G_{sse}, A_j^2, A_j^3 and a_j already have neighboring check nodes of degree 1.
2) $A_j^1 \notin F_3$, $A_j^2 \notin F_3$, and $A_j^3 \notin F_3$. In this case, we can replace a_j by A_j^1 in F_3, and still get a one-iteration elimination set, for the same reason as the above case (and the fact that more edges outside U_j^1 will be removed by this replacement because A_j^1 is an "interface node" and a_j is not).

So from F_3, we can obtain a new one-iteration elimination set F_4 of at most α nodes, where

$$F_4 \cap \{y_i \mid 1 \le i \le n\} = \emptyset,$$
$$F_4 \cap \{p_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset,$$
$$F_4 \cap \{q_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset$$
$$F_4 \cap \{d_j^t \mid 1 \le j \le k,\ 1 \le t \le 6\} = \emptyset,$$
$$F_4 \cap \{a_j \mid 1 \le j \le k\} = \emptyset,$$

and

$$F_4 \cap \{b_j \mid 1 \le j \le k\} = \emptyset.$$

Let $\hat{F} = F_4$. Clearly, $\hat{F} \subseteq I_{sse}$. That concludes the proof. ∎

Some properties of Canonical Elimination Sets are shown in the next lemma. We first define "endpoints of an H bar."

Definition 22. *Let u be any node in*

$$\{A_j^t \mid 1 \le j \le k, 1 \le t \le 3\} \cup \{B_j^t \mid 1 \le j \le k, 1 \le t \le 3\},$$

and let v be any node in

$$\{X_i^1 \mid 1 \le i \le n\} \cup \{X_i^0 \mid 1 \le i \le n\}.$$

If u and v are connected by an H bar, then they are called the two endpoints of that H bar.

Example 23. *In Fig. 8 (f), the endpoints of H bars are (A_j^t, X_i^1) and (B_j^t, X_i^0). In In Fig. 8 (g), such endpoint pairs are (A_j^t, X_i^0) and (B_j^t, X_i^1).* □

Lemma 24. *For the graph G_{sse}, a Canonical Elimination Set F has the following properties:*

- *Property 1: For $i = 1, 2, \cdots, n$, either*
$$X_i^1 \in F$$
or
$$X_i^0 \in F.$$
- *Property 2: For $j = 1, 2, \cdots, k$ and $t = 1, 2, 3$, either*
$$A_j^t \in F$$
or
$$B_j^t \in F.$$
- *Property 3: For $j = 1, 2, \cdots, k$,*
$$|F \cap \{A_j^1, A_j^2, A_j^3\}| \ge 1$$
and
$$|F \cap \{B_j^1, B_j^2, B_j^3\}| \ge 1.$$
- *Property 4: If u and v are the two endpoints of an H bar, then either*
$$u \in F$$
or
$$v \in F.$$

Proof: For the gadget V_i (see Fig. 8 (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.

For the gadget W_j (see Fig. 8 (c)), for the pair (A_j^1, B_j^1), if neither A_j^1 nor B_j^1 is in F, then the BP algorithm cannot decode d_j^1 in one iteration since both of d_j^1's neighboring check nodes will have degree 2. The similar analysis holds for the pairs (A_j^2, B_j^2) and (A_j^3, B_j^3). So Property 2 is true.

For the gadget U_j^1 (see Fig. 8 (b)), if none of A_j^1, A_j^2, A_j^3 is in F, then the BP algorithm cannot decode a_j in one iteration since all of d_j^1's three neighboring check nodes will have degree 2. The same analysis holds for U_j^2. So Property 3 is true.

For the H bar (see Fig. 8 (f) and (g)) between u and v, if neither u nor v is in F, then the variable node in the H bar – which is labelled as p_j^t or q_j^t in Fig. 8 (f) and (g) – cannot be decoded by the BP algorithm in one iteration because both of its neighboring check node will have degree 2. So Property 4 is true. ∎

Corollary 25. *If F is a One-Iteration Elimination Set of G_{sse}, then*

$$|F| \ge n + 3k.$$

Proof: If F is a Canonical Elimination Set, by Property 1 and Property 2 in Lemma 24, we get $|F| \ge n + 3k$. Then by Lemma 21, the same conclusion holds for any One-Iteration Elimination Set. ∎

Definition 26. *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}.

Lemma 27. *An Ideal Elimination Set F of G_{sse} has these properties:*

- *Property 1: For $i = 1, 2, \cdots, n$, either X_i^1 or X_i^0 is in F, but not both.*
- *Property 2: For $j = 1, 2, \cdots, k$ and $t = 1, 2, 3$, either A_j^t or B_j^t is in F, but not both.*
- *Property 3: For $j = 1, 2, \cdots, k$, 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 $\{B_j^1, B_j^2, B_j^3\}$.*
- *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.*

Proof: Given that $|F| = n + 3k$, Properties 1, 2 and 3 here follow directly from Properties 1, 2 and 3 in Lemma 24.

Now consider Property 4 here. WLOG, suppose that $u = A_j^t$ and $v = X_i^1$ for some i, j, t. (The other cases can be analyzed similarly because of symmetry.) Consider two cases:

1) $u \in F$. In this case, by Property 2, $B_j^t \notin F$. By the construction of G_{sse}, B_j^t and X_i^0 are the two endpoints of another H bar. So by Property 4 in Lemma 24, since $B_j^t \notin F$, we get $X_i^0 \in F$. Then by Property 1, we have $v = X_i^1 \notin F$.
2) $u \notin F$. In this case, by Property 4 in Lemma 24, we have $v = X_i^1 \in F$.

Therefore $u \in F$ if and only if $v \notin F$. So Property 4 is true. ∎

Given an Ideal Elimination Set of G_{sse}, we can construct a solution to the Not-all-equal SAT Problem as follows.

Definition 28. *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 $X_i^1 \in F$.*

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

Lemma 29. *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

$$A_j^t \notin F.$$

Proof: Let l_j^t denote the t-th literal in the clause C_j. Consider two cases:

- Case 1: l_j^t is x_i for some $1 \leq i \leq n$. In this case, by the construction of G_{sse}, A_j^t is connected to X_i^1 by an H bar. l_j^t is true if and only if $X_i^1 \in F$, which – by Property 4 of Lemma 27 – happens if and only if $A_j^t \notin F$.
- Case 2: l_j^t is $\bar{x}_i$ for some $1 \leq i \leq n$. In this case, by the construction of G_{sse}, A_j^t is connected to X_i^0 by an H bar. l_j^t is true if and only if x_i is false, which happens if and only if $X_i^0 \in F$, which – by Property 4 of Lemma 27 – happens if and only if $A_j^t \notin F$.

So in both cases, the conclusion holds. ∎

Lemma 30. *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 $A_j^{t_1} \in F$ and $A_j^{t_2} \notin F$. By Property 3 of Lemma 27, such two integers $t_1, t_2 \in \{1, 2, 3\}$ exist. Consider the clause C_j. By Lemma 29, 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 31. *Let π be a satisfying solution to the Not-all-equal SAT Problem; that is, with the solution π, every clause has at least one true literal and at least one false literal. A corresponding set of nodes, $\mathcal{F}(\pi)$, in G_{sse} is constructed as follows:*

- *For $i = 1, 2, \cdots, n$, if $x_i = 1$ in the solution π, then $X_i^1 \in \mathcal{F}(\pi)$ and $X_i^0 \notin \mathcal{F}(\pi)$; otherwise, $X_i^1 \notin \mathcal{F}(\pi)$ and $X_i^0 \in \mathcal{F}(\pi)$.*
- *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 π, then $A_j^t \notin \mathcal{F}(\pi)$ and $B_j^t \in \mathcal{F}(\pi)$; otherwise, $A_j^t \in \mathcal{F}(\pi)$ and $B_j^t \notin \mathcal{F}(\pi)$.*

Lemma 32. *Let π be a satisfying solution to the Not-all-equal SAT Problem. Then $\mathcal{F}(\pi)$ is an Ideal Elimination Set of G_{sse}.*

Proof: Let us first show that $\mathcal{F}(\pi)$ is an One-Iteration Elimination Set of G_{sse}. Consider nodes in the following gadgets of G_{sse}:

- Consider the gadget V_i, for $1 \leq i \leq n$. (See Fig. 8 (a).) By the construction of $\mathcal{F}(\pi)$, either X_i^1 or X_i^0 is in $\mathcal{F}(\pi)$. Either way, after the node in $\mathcal{F}(\pi)$ is removed, the other two variable nodes in V_i will have neighboring check nodes of degree 1.
- Consider the gadget W_j, for $1 \leq j \leq k$. (See Fig. 8 (c).) Since the clause C_j has at least one true literal and at least one false literal, WLOG, let us suppose that its 1st and 2nd literals are true, and its 3rd literal is false. (There are totally 6 cases. And the other 5 cases can be proved similarly by symmetry.) By the construction of $\mathcal{F}(\pi)$, we have $A_j^1 \notin \mathcal{F}(\pi)$, $B_j^1 \in \mathcal{F}(\pi)$, $A_j^2 \notin \mathcal{F}(\pi)$, $B_j^2 \in \mathcal{F}(\pi)$, $A_j^3 \in \mathcal{F}(\pi)$, $B_j^3 \notin \mathcal{F}(\pi)$. It is not hard to see that after B_j^1, B_j^2 and B_j^3 are removed, the remaining

variable nodes in W_j will all have neighboring check nodes of degree 1: removing A_j^3 will cause that effect for nodes in U_j^1, removing B_j^1 (and B_j^2) will cause that effect for nodes in U_j^2, and removing all those three nodes will cause that effect for d_j^1, d_j^2, $\cdots$, d_j^6.

- Consider an H bar. (See Fig. 8 (f) and (g).) Let u and v be the two endpoints of the H bar, where u is A_j^t or B_j^t, and v is X_i^1 or X_i^0 (for some parameters i, j, t). There are four possible cases:
  - Case 1: u is A_j^t and v is X_i^1. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is x_i. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^1 \in \mathcal{F}(\pi)$ and $A_j^t \notin \mathcal{F}(\pi)$; otherwise, $X_i^1 \notin \mathcal{F}(\pi)$ and $A_j^t \in \mathcal{F}(\pi)$.
  - Case 2: u is A_j^t and v is X_i^0. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is $\bar{x}_i$. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^0 \notin \mathcal{F}(\pi)$ and $A_j^t \in \mathcal{F}(\pi)$; otherwise, $X_i^0 \in \mathcal{F}(\pi)$ and $A_j^t \notin \mathcal{F}(\pi)$.
  - Case 3: u is B_j^t and v is X_i^1. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is $\bar{x}_i$. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^1 \in \mathcal{F}(\pi)$ and $B_j^t \notin \mathcal{F}(\pi)$; otherwise, $X_i^1 \notin \mathcal{F}(\pi)$ and $B_j^t \in \mathcal{F}(\pi)$.
  - Case 4: u is B_j^t and v is X_i^0. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is x_i. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^0 \notin \mathcal{F}(\pi)$ and $B_j^t \in \mathcal{F}(\pi)$; otherwise, $X_i^0 \in \mathcal{F}(\pi)$ and $B_j^t \notin \mathcal{F}(\pi)$.

  So in all four cases, either u or v is in $\mathcal{F}(\pi)$. If $u \in \mathcal{F}(\pi)$, then after u is removed from G_{sse}, both v and the variable node in the H bar (p_j^t or q_j^t) will have a neighboring check node of degree 1; and the same holds if $v \in \mathcal{F}(\pi)$.

So after nodes in $\mathcal{F}(\pi)$ are removed, the remaining variable nodes in G_{sse} will all have neighboring check nodes of degree 1. So $\mathcal{F}(\pi)$ is an One-Iteration Elimination Set of G_{sse}. Then it can be seen that all the nodes in $\mathcal{F}(\pi)$ are Interface Nodes (namely, $\mathcal{F}(\pi) \subset I_{sse}$) and $|\mathcal{F}(\pi)| = n + 3k$. So $\mathcal{F}(\pi)$ is an Ideal Elimination Set of G_{sse}. ■

We now prove the NP-hardness of the SSE_1 Problem.

Theorem 33. *The SSE_1 Problem is NP-hard.*

Proof: The SSE_1 Problem is an optimization Problem. Consider its decision problem: "Given a Stopping Graph G and an integer t, is it possible to remove t variable nodes from G so that the BP algorithm can decode the remaining variable nodes in one iteration?" Let it be called the "$SSE_1^{decision}$ Problem." Clearly, the problem is in NP. We will show now that there is a polynomial-time reduction from the NP-complete Not-all-equal SAT Problem to the $SSE_1^{decision}$ Problem.

Corresponding to the Not-all-equal SAT Problem, it has been introduced how to construct a bipartite graph G_{sse}. Let G be G_{sse}, and let t be $n+3k$. Then we have a mapping from the Not-all-equal SAT Problem to the $SSE_1^{decision}$ Problem. It is not difficult to see that the mapping tokes polynomial time. We now need to prove that the Not-all-equal SAT Problem is satisfied if and only if the corresponding $SSE_1^{decision}$ Problem has a positive answer:

1) If the Not-all-equal SAT Problem is satisfiable, let π be such a satisfying solution. By Lemma 32, $\mathcal{F}(\pi)$ is an Ideal Elimination Set of G_{sse}, which has size $n + 3k$. So the $SSE_1^{decision}$ Problem has a positive answer.
2) If the $SSE_1^{decision}$ Problem has a positive answer, then G_{sse} has an One-Iteration Elimination Set of size $n + 3k$. By Lemma 21 and Corollary 25, G_{sse} has a Canonical Elimination Set F of size $n + 3k$, which, by Definition 26, is also an Ideal Elimination Set. Then by Lemma 30, the Not-all-equal SAT Problem is satisfiable.

So there is a polynomial-time reduction from the Not-all-equal SAT Problem to the $SSE_1^{decision}$ Problem. So the $SSE_1^{decision}$ Problem is NP-complete, and the SSE_1 Problem is NP-hard. ■

V. Approximation Algorithm for SSE_1 Problem

In this section, we present an approximation algorithm for the SSE_1 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 34. *In the Stopping Graph $G = (V \cup C, E)$, $\forall\, v \in V$, define its "variable-node neighborhood" as $\Lambda(v) \triangleq$*

$$\{u \in V - \{v\} \mid \exists\, c \in C \text{ such that } (u, c) \in E \text{ and } (v, c) \in E\}.$$

That is, every variable node in $\Lambda(v)$ shares a common neighboring check node with v.

Example 35. *For the Stopping Graph in Fig.* 10*, we have $\Lambda(v_1) = \{v_4, v_6\}$, $\Lambda(v_2) = \{v_3, v_8\}$, $\Lambda(v_3) = \{v_2, v_5, v_8\}$, and so on.* □

We now introduce an approximation algorithm. 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' color turns *black*, it means the algorithm has included it in the Elimination Set (namely, the algorithm has removed it).

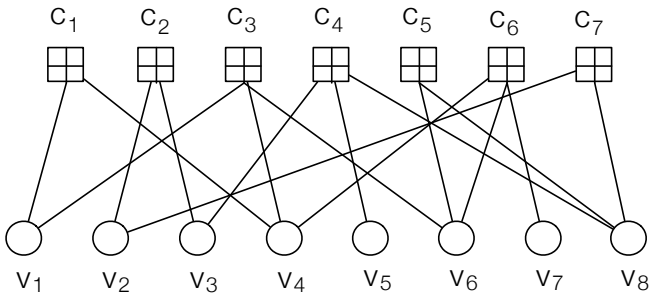

Fig. 10. A Stopping Graph $G = (V \cup C,\ E)$.

- 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.

When the algorithm ends, every variable node's color will be either *black* or *gray*.

The algorithm works as follows. It takes a greedy approach, and updates the colors of variable nodes iteratively. In each iteration, it identifies an arbitrary *white* variable node (say it is node v), and does the following:

1) Step One: 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*.
2) Step Two: 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*.

The algorithm keeps iterating as above until all variable nodes are either *black* or *gray*. Then it returns the set of *black* variable nodes as the Elimination Set. It is not difficult to see that the remaining variable nodes are decodable by BP in one iteration: by the algorithm, every time a variable node is turned from *white* to *gray*, it has at least one neighboring check node c such that c' other neighboring variable nodes are all *black* (namely, removed), and an BP iteration using the check node c will help decode that *gray* variable node.

The algorithm is formally presented below. Note that it uses two sets, S_{white} and S_{black}, to keep track of the white and black nodes, respectively.

Algorithm 36 *Approximation Algorithm for SSE_1*
Input: Stopping Graph $G = (V \cup C, E)$.
Output: A one-iteration elimination set.
Algorithm:

1) $S_{white} \leftarrow V$, $S_{black} \leftarrow \emptyset$.
2) **for** $v \in V$
3) $\quad color(v) \leftarrow white$.
4) **while** $S_{white} \neq \emptyset$
5) {
6) *Pick an arbitrary node v from S_{white}.*
7) $U_v \leftarrow \emptyset$.
8) **for** $u \in \Lambda(v)$
9) {
10) **if** $color(u) = white$
11) {
12) $U_v \leftarrow U_v \cup \{u\}$.
13) $color(u) \leftarrow black$.
14) $S_{black} \leftarrow S_{black} \cup \{u\}$.
15) $S_{white} \leftarrow S_{white} - \{u\}$.
16) }
17) **else if** $color(u) = gray$
18) {
19) $U_v \leftarrow U_v \cup \{u\}$.
20) $color(u) \leftarrow black$.
21) $S_{black} \leftarrow S_{black} \cup \{u\}$.
22) }
23) }
24) $S_{white} \leftarrow S_{white} - \{v\}$, $color(v) \leftarrow gray$.
25) **for** $u \in U_v$
26) **for** *every check node c adjacent to u*
27) **if** *exactly one of c's neighboring nodes is white and all c's other neighboring nodes are black*
28) {
29) *Let w be that white neighboring node.*
30) $S_{white} \leftarrow S_{white} - \{w\}$, $color(w) \leftarrow gray$.
31) }
32) }
33) **return** S_{black}.

We show an example of the above algorithm.

Example 37. *Let the graph G be as shown in Fig. 11 (a). The algorithm first identifies a white node v_1, turns nodes in $U_{v_1} = \{v_2, v_3\}$ black, then turns v_1 and v_4 gray (see Fig. 11 (b)). Next, it identifies a white node v_5, turns nodes in $U_{v_5} = \{v_6, v_7\}$ black, then turns v_5 gray (see Fig. 11 (c)). Next, it identifies a white node v_8, turns the nodes in $U_{v_8} = \{v_9\}$ black, then turns v_8 gray (see Fig. 11 (d)). So the Elimination Set is $S = \{v_2, v_3, v_6, v_7, v_9\}$. If we delete nodes in S from G, we get the subgraph in Fig. 11 (e), where we can see that all the remaining nodes are gray and can be decoded by one BP iteration.*

It is not hard to verify that every one-iteration elimination set for G has size no less than 5. Since $|S| = 5$, the output of the algorithm is actually optimal in this example. □

We analyze the time complexity of the above algorithm. Let d_v and d_c denote the maximum degrees of variable nodes and check nodes in the Stopping Graph G, respectively. The algorithm has time complexity $O(d_v^2 d_c^2 |V|)$ because it identifies up to $O(|V|)$ white variable nodes and for each such node v, it checks its neighboring check nodes and nodes in $\Lambda(v)$, check nodes adjacent to nodes in $\Lambda(v)$, and variable nodes that share common neighboring check nodes with any node in $\Lambda(v)$; and there are $O(d_v^2 d_c^2)$ such nodes for each v.

We now analyze the approximation ratio of the algorithm. We first introduce a few lemmas.

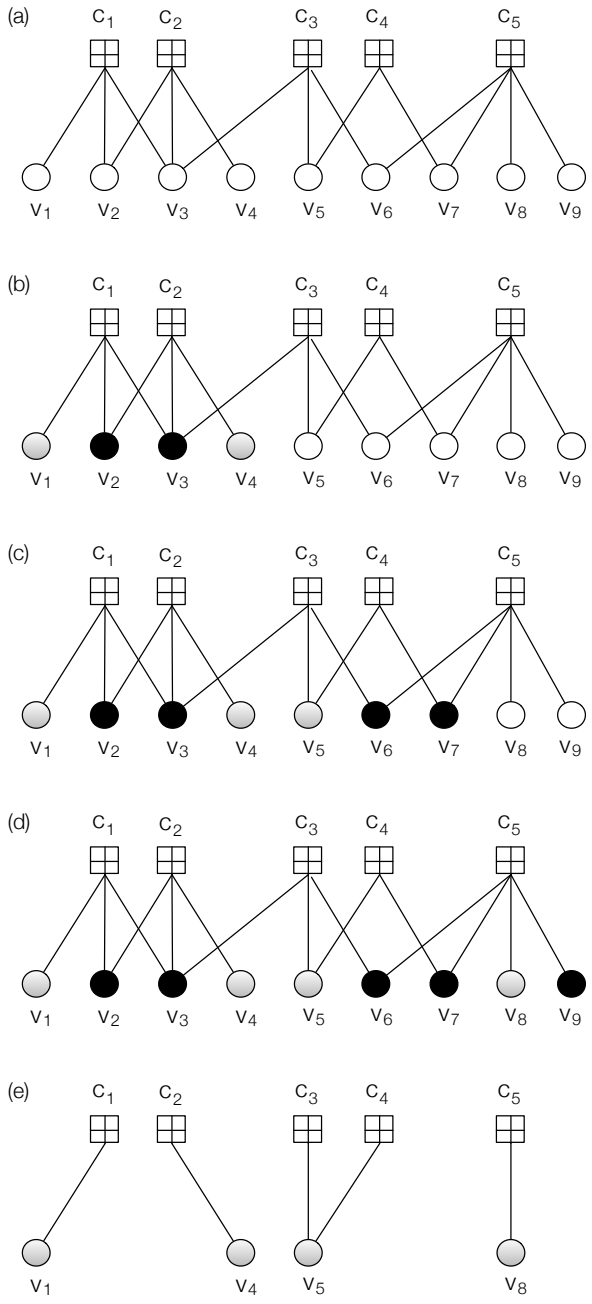

Fig. 11. An example of the approximation algorithm for SSE_1.

Lemma 38. *Let $G_1 = (V_1 \cup C_1,\ E_1)$ and $G_2 = (V_2 \cup C_2,\ E_2)$ be two Stopping Graphs. Let s_1 and s_2 denote the size of the minimum one-iteration elimination set in G_1 and G_2, respectively. If G_2 can be obtained from G_1 by removing some variable nodes and their incident edges, then*

$$s_2 \leq s_1.$$

Proof: Removing variable nodes is the same as knowing the values of those erased codeword bits, which only helps BP decoding. ■

Say that the algorithm identifies a sequence of white variable nodes

$$\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_t$$

in the Stopping Graph $G = (V \cup C, E)$, and turns the variable nodes in

$$U_{\hat{v}_1}, U_{\hat{v}_2}, \cdots, U_{\hat{v}_t}$$

black. Let us define a sequence of subgraphs

$$G_0, G_1, \cdots, G_t$$

accordingly.

Definition 39. *Let $G_0 = G$. For $i = 1, 2, \cdots, t$, let G_i be obtained from G_{i-1} by removing the nodes in*

$$U_{\hat{v}_i} \cup \{\hat{v}_i\} \cup \{\textit{check nodes adjacent to } \hat{v}_i\}$$

and their incident edges.

Note that for $i = 1, 2, \cdots, t$, in the i-th iteration, the algorithm removes only the variable nodes in $U_{\hat{v}_i}$ (namely, turning them black) from the subgraph G_{i-1}, not $\hat{v}_i$ or its adjacent check nodes. (It turns $\hat{v}_i$ to gray.) However, once $U_{\hat{v}_i}$ is removed, all the nodes in $\Lambda(\hat{v}_i)$ are removed, so $\hat{v}_i$ 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_1 Problem for G_i in the next iteration, and it can be seen that

$$\hat{v}_{i+1}, U_{\hat{v}_{i+1}}, \{\text{check nodes adjacent to } \hat{v}_{i+1}\},$$

$$\hat{v}_{i+2}, U_{\hat{v}_{i+2}}, \{\text{check nodes adjacent to } \hat{v}_{i+2}\},$$

$$\cdots,$$

$$\hat{v}_t, U_{\hat{v}_t}, \{\text{check nodes adjacent to } \hat{v}_t\}$$

are all nodes (or sets of nodes) in G_i.

Lemma 40. *For $i = 0, 1, \cdots, t-1$, every one-iteration elimination set for G_i contains at least one variable node in*

$$U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}.$$

Proof: Consider the graph G_i and its variable node $\hat{v}_{i+1}$. To make the remaining variable nodes decodable by one iteration of BP decoding after some variable nodes are removed, it is *necessary* (although not sufficient) that $\hat{v}_{i+1}$ is either removed, or decodable after one such iteration of BP decoding; and that requires one of these two conditions to be true:

1) $\hat{v}_{i+1}$ is removed.
2) $\hat{v}_{i+1}$ is not removed, but it has a neighboring check node c such that all c's other neighboring variable nodes in G_i are removed. (The check node c will help decode $\hat{v}_{i+1}$ in one iteration of BP decoding. And note that those "other neighboring variable nodes" of c are all nodes in $U_{\hat{v}_{i+1}}$. Also note that since v_{i+1} is turned from white to gray in the $(i+1)$-th iteration of the algorithm, at the

beginning of the $(i+1)$-th iteration, every check node adjacent to v_{i+1} must have degree at least two in G_i. So the set of those "other neighboring variable nodes" of c cannot be empty.)

So it is necessary that at least one variable node in $U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}$ is removed. ∎

Lemma 41. *For $i = 0, 1, \cdots, t$, let α_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 $i = t, t-1 \cdots$ down to 0). When $i = t$, clearly $\alpha_i \geq t - i = 0$, so the conclusion holds for the base case. Now assume that the conclusion holds for $\alpha_t, \alpha_{t-1}, \cdots, \alpha_{i+1}$, and consider the case for α_i.

Consider an optimal (i.e., minimum-sized) one-iteration elimination set S for G_i. Define $Y \triangleq S \cap (U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\})$. By Lemma 40, S removes at least one variable node in $U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}$, so $|Y| \geq 1$. Let $\tilde{G}$ be the bipartite graph obtained by removing the variable nodes in Y from G_i (and their incident edges), and let $\tilde{\alpha}$ denote the minimum size of a one-iteration elimination set for $\tilde{G}$. Then $\alpha_i = |S| = |Y| + \tilde{\alpha} \geq \tilde{\alpha} + 1$.

G_{i+1} is obtained from G_i by removing the variable nodes in $U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}$, which is a superset of Y. So G_{i+1} can also be obtained from $\tilde{G}$ by removing the variable nodes in $(U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}) - Y$. So by Lemma 38, $\alpha_{i+1} \leq \tilde{\alpha}$. By the induction assumption, we get $\alpha_{i+1} \geq t-(i+1)$. By combining the above results, we get $\alpha_i \geq \tilde{\alpha} + 1 \geq \alpha_{i+1} + 1 \geq t - (i+1) + 1 = t - i$. ∎

Theorem 42. *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 41, 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

$$U_{\hat{v}_1} \cup U_{\hat{v}_2} \cup \cdots \cup U_{\hat{v}_t},$$

whose size is

$$|\bigcup_{i=1}^{t} U_{\hat{v}_i}| = \sum_{i=1}^{t} |U_{\hat{v}_i}| \leq \sum_{i=1}^{t} |\Lambda(\hat{v}_i)| \leq t \cdot d_v(d_c - 1).$$

So the approximation ratio is at most $d_v(d_c - 1)$. ∎

VI. Analysis and Algorithms for SSE_k Problems

In this section, we present more analysis and algorithms for SSE_k Problems, including $k = \infty$. We first analyze how an important factor, RBER (raw bit-erasure rate), affects the performance of approximation algorithms, and show that for high-rate codes with high actual erasure rates, all algorithms have good approximation ratios. We then present exact algorithms for SSE_∞ and SSE_k Problems when the Stopping Graph is a tree (or a forest). The algorithms output optimal solutions and have linear time complexity.

A. Effect of RBER for Approximation Algorithms

We first analyze the effect of RBER for approximation algorithms. 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. As shown in Fig. 3 (b), 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 43. *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 decoded by the NR-Decoder, the total number of codeword bits with known values is $(N-|V|)+|S| < (N-|V|)+(|V|-N+K) = K$. Then the ECC-Decoder will not be able to recover the K bits of information in the codeword. ∎

Theorem 44. *For the SSE_k Problem, if $\epsilon > 1 - R$, then 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 43, $|S^*| \geq |V| - N + K$. 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.

B. Exact Algorithm for SSE_∞ Problem with Stopping Tree

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.

In this subsection, we present an efficient and exact algorithm for the SSE_∞ Problem. The algorithm will be extended to the SSE_k Problem for general k subsequently.

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, $\cdots$, $v_{|V|+|C|}$ 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}.

The algorithm for SSE_∞ is as follows.

Algorithm 45 *Exact Algorithm for SSE_∞*
Input: Stopping Tree $G = (V \cup C, E)$.
Output: An Elimination Set of minimum size in G.
Algorithm:

1) *Generate G_{BFS} by running BFS on G.*
2) *Let S be an empty set.*
3) $i \leftarrow |V| + |C|$.
4) **while** $i \geq 1$
5) {
6) **if** $i > 1$ *and* $v_i \in V$
7) {
8) *Let j be the minimum index such that v_j is a sibling*
9) *of v_i in the BFS tree G_{BFS} rooted at v_1.*
10) **if** $j < i$
11) *Add $v_j, v_{j+1}, \cdots, v_{i-1}$ to set S.*
12) $i \leftarrow j - 1$.
13) }
14) **else if** $i = 1$
15) {
16) *Add v_1 to set S.*
17) $i = 0$.
18) }
19) **else**
20) $i \leftarrow i - 1$.
21) }
22) **Return** S.

The algorithm first runs BFS to generate G_{BFS}. It then processes the nodes in the reverse order of their labels (from $v_{|V|+|C|}$ to v_1). Every time it comes to a node v_i, if v_i is a variable node and has siblings (of smaller labels), its siblings are included in the Elimination Set. The root v_1 is also included in the Elimination Set. The following is an example of the algorithm.

Example 46. *A Stopping Tree and its BFS tree are shown in Fig.* 12 *(a) and (b), respectively. (Note that the node labels v_1, v_2, $\cdots$, v_{17} in Fig.* 12 *(a) are not known* a priori*; instead, they are obtained after we run BFS on the graph with v_1 as its root.) Then algorithm then processes the nodes in the reverse order of their labels. (Note that when it comes to a check node, no action is taken.) It first comes to v_{17}, and includes its sibling v_{16} in the Elimination Set. (See Fig.* 12 *(c).) It then comes to v_{15}, and includes its siblings v_{14} and v_{13} in the Elimination Set. (See Fig.* 12 *(d).) It then comes to v_{12}, and takes no action since v_{12} has no sibling. (See Fig.* 12 *(e).) It then comes to v_{11}, v_{10} and v_9 sequentially and takes no action since they are check nodes. It then comes to v_8, and includes its sibling v_7 in the Elimination Set. (See Fig.* 12 *(f).) It then comes to v_6, v_5, $\cdots$, v_2 sequentially and takes no action since they either have no sibling or are check nodes. Finally, it comes to the root v_1 and includes it in the Elimination Set. (See Fig.* 12 *(i).) The Elimination Set returned by the algorithm is $\{v_1, v_7, v_{13}, v_{14}, v_{16}\}$.* □

We now show that the algorithm returns an optimal (i.e.,

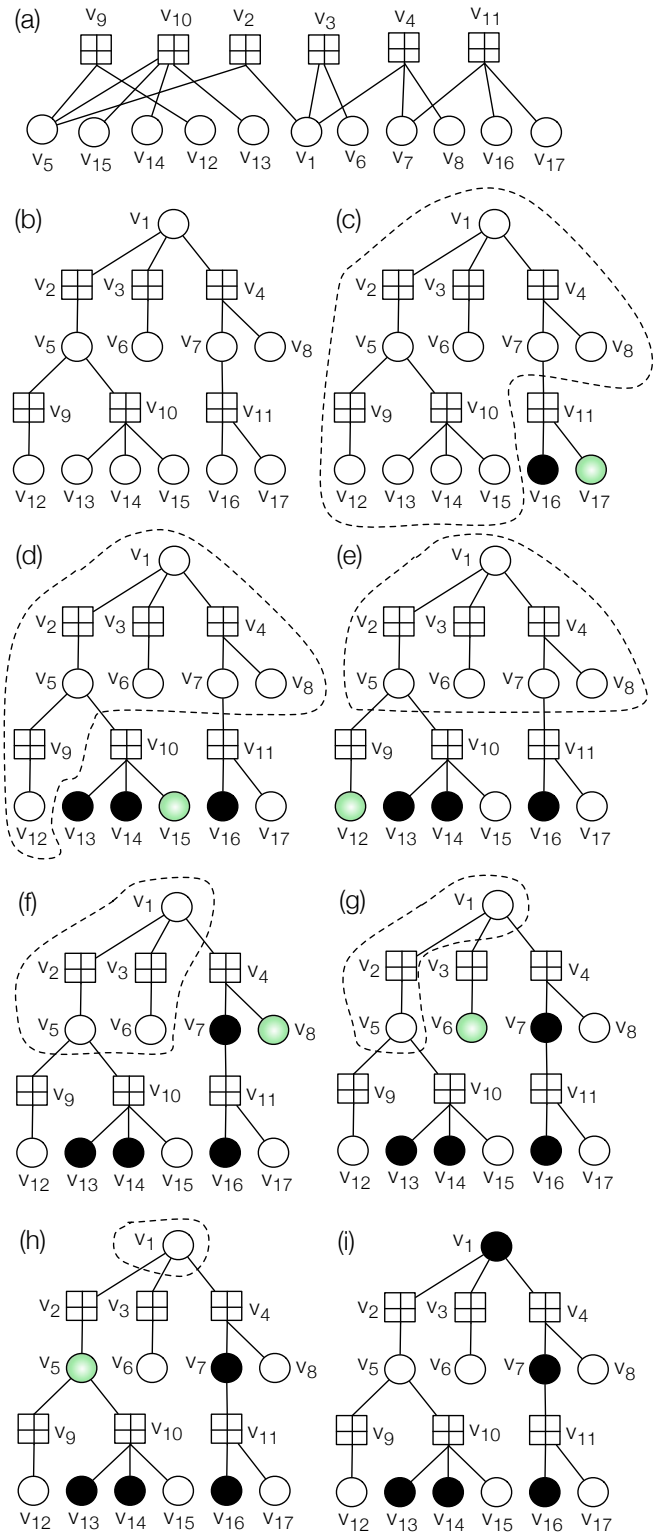

Fig. 12. Algorithm for SSE_∞ on a Stopping Tree. (a) A Stopping Tree $G = (V \cup C, E)$. (b) Its BFS (Breadth-First Search) tree G_{BFS}. (c) Process v_{17}. (d) Process v_{15}. (e) Process v_{12}. (f) Process v_8. (g) Process v_6. (h) Process v_5. (i) Process v_1.

minimum-sized) Elimination Set. We suppose $|V| \geq 2$. (The case of $|V| = 1$ is trivial.)

Lemma 47. *In G_{BFS}, let B denote the set of sibling nodes of $v_{|V|+|C|}$. (B could be empty.) Then any Elimination Set for G_{BFS} contains at least $|B|$ nodes in $B \cup \{v_{|V|+|C|}\}$.*

Proof: If B is an empty set, the conclusion is clearly true. So now assume B is not empty. Let c be the parent of $v_{|V|+|C|}$ (and also all the nodes in B) in G_{BFS}, which is a check node. Let w be the parent of c, which is a variable node. We can see that c is adjacent to $|B| + 2$ variable nodes in total.

Since $v_{|V|+|C|}$ has the greatest label among all nodes, by the property of BFS, it has the greatest distance to the root among all nodes; and so do its siblings. So all nodes in $B \cup \{v_{|V|+|C|}\}$ are leaves in G_{BFS}.

We now prove the lemma by contradiction. If fewer than $|B|$ nodes in $B \cup \{v_{|V|+|C|}\}$ are included in an Elimination Set, then at least two leaves in $B \cup \{v_{|V|+|C|}\}$ – say v_i and v_j – are not in the Elimination Set. Since c is the only check node adjacent to them, even if other nodes of the tree are all in the Elimination Set, v_i and v_j still cannot be decoded. And that contradicts the definition of Elimination Set. So the conclusion is true. ∎

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}. (For instance, in Fig. 12, we have $v_{|V|+|C|} = v_{17}$, $\pi(v_{17}) = v_{11}$, and G_{sub} is the subtree in the dashed circle in Fig. 12 (c).)

Lemma 48. *Let S^* be a minimum-sized Elimination Set of G_{sub}, and let B denote the set of sibling nodes of $v_{|V|+|C|}$. Then $S^* \cup B$ is a minimum-sized Elimination Set of G_{BFS}.*

Proof: Let us call an Elimination Set *normalized* if it includes all the nodes in B but does not include $v_{|V|+|C|}$. By Lemma 47, an Elimination Set of G_{BFS} contains either $|B|$ or $|B|+1$ nodes of $B \cup \{v_{|V|+|C|}\}$. If it is the latter case, by replacing $\{v_{|V|+|C|}\}$ by $\pi(\pi(v_{|V|+|C|}))$ in the Elimination Set (which makes all the variable nodes adjacent to $\pi(v_{|V|+|C|})$ except $v_{|V|+|C|}$ included in the Elimination Set), we can get another Elimination Set of no greater size. Therefore, there exists a minimum-sized Elimination Set that is normalized.

Consider a normalized Elimination Set $\tilde{S} \subseteq V$. Since $B \subseteq \tilde{S}$ and $v_{|V|+|C|} \notin \tilde{S}$, $\tilde{S} - B$ must be an Elimination Set of G_{sub}. On the other hand, given an Elimination Set $\hat{S}$ of G_{sub}, $\hat{S} \cup B$ must be a normalized Elimination Set of G_{BFS}. (The BP decoding algorithm will first decode all the variable nodes in G_{sub}, then use the check node $\pi(v_{|V|+|C|})$ to decode $v_{|V|+|C|}$.) Since S^* is a minimum-sized Elimination Set of G_{sub}, $S^* \cup B$, as a normalized Elimination Set for G_{BFS}, is minimum-sized. Since there exists a minimum-sized Elimination Set that is normalized, $S^* \cup B$, as an Elimination Set for G_{BFS} (without the restriction of being normalized), is also minimum-sized. ∎

Theorem 49. *Algorithm 45 returns an optimal (i.e., minimum-sized) Elimination Set of $G = (V \cup C, E)$.*

Proof: By Lemma 48, the problem of finding an optimal Elimination Set for G_{BFS} (which is the same as G) can be reduced to the problem of finding an optimal Elimination Set for its subtree G_{sub}. Algorithm 45 uses that technique repeatedly to reduce the problem to smaller and smaller subtrees, until it comes to the final case where the subtree contains only the root node v_1 (whose optimal Elimination Set is simply $\{v_1\}$). (In Fig. 12, such a sequence of shrinking subtrees are shown in dashed circles from (c) to (h).) That leads to the conclusion. ∎

Therefore Algorithm 45 is an exact algorithm for the SSE_∞ Problem. Its time complexity is $O(|V| + |C|)$.

C. Exact Algorithm for SSE_k Problem with Stopping Tree

We now extend the previous analysis, and design an exact algorithm for the SSE_k Problem of linear time complexity.

The algorithm first runs BFS on G to get the tree G_{BFS} that labels nodes by $v_1, v_2, \cdots, v_{|V|+|C|}$, where v_1 is the root. Then (similar to the algorithm for SSE_∞), 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 $gSSE_k$ Problem – to smaller and smaller subtrees. Let us now define this $gSSE_k$ Problem.

Definition 50. *[$gSSE_k$ 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*

$$\delta(v) \in \{1, 2, \cdots, k, \infty\}$$

and

$$\omega(v) \in \{0, 1, \cdots, k, \infty\}$$

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-erasure) 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 $\tau \in \{1, 2, \cdots, |V|+|C|\}$ denote the minimum integer such that v_τ either is a sibling of $v_{|V|+|C|}$ or is $v_{|V|+|C|}$ itself. (So $v_\tau, v_{\tau+1}, \cdots, v_{|V|+|C|}$ are siblings.) Define

$$\mathcal{P} \triangleq \{i \mid \tau \leq i \leq |V| + |C|, \omega(v_i) \leq k\}$$

and

$$\mathcal{Q} \triangleq \{i \mid \tau \leq i \leq |V| + |C|, \delta(v_i) \leq k\}.$$

Since $\forall\ v \in V$, either $\delta(v)$ or $\omega(v)$ is ∞ but not both, $\mathcal{P}$ and $\mathcal{Q}$ form a partition of the set $\{\tau, \tau+1, \cdots, |V|+|C|\}$.

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 51. *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 $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ but not v_{i^*}.*

Proof: Any g-Elimination Set for G_{BFS} has to include at least $|\mathcal{P}| - 1$ nodes in $\{v_i | i \in \mathcal{P}\}$ because otherwise the un-included nodes in $\{v_i | i \in \mathcal{P}\}$ will never be corrected. It is an optimal strategy to include the $|\mathcal{P}| - 1$ nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ in the g-Elimination Set because their $\omega(v_i)$ values impose more restrictive requirements than $\omega(v_{i^*})$ does. Now let T be a g-Elimination Set for G_{BFS} that includes all the nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$. If $v_{i^*} \in T$, we can replace it by $\pi(\pi(v_{|V|+|C|}))$ in T and get another g-Elimination Set T' for G_{BFS}, with $|T'| \leq |T|$ (since $\pi(\pi(v_{|V|+|C|}))$ may already be in T). (Note that with T', since $\max_{i\in\mathcal{P}} \omega(v_i) > \max_{i\in\mathcal{Q}} \delta(v_i)$, the check node $\pi(v_{|V|+|C|})$ can help correct v_{i^*} by iteration $\max_{i\in\mathcal{Q}} \delta(v_i) + 1 \leq \max_{i\in\mathcal{P}} \omega(v_i) = \omega(v_{i^*})$; and since $\pi(\pi(v_{|V|+|C|})) \in T'$, the BP decoding in G_{sub} becomes independent of the subtree rooted at $\pi(v_{|V|+|C|})$.) So there exists a minimum-sized g-Elimination Set for G_{BFS} that includes the nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ but not v_{i^*}. ∎

Lemma 52. *Suppose $\max_{i\in\mathcal{P}} \omega(v_i) \leq \max_{i\in\mathcal{Q}} \delta(v_i)$. Then there exists a minimum-sized g-Elimination Set for G_{BFS} that contains all the nodes in $\{v_i | i \in \mathcal{P}\}$.*

Proof: If $\mathcal{P} = \emptyset$, the conclusion automatically holds. If $\mathcal{P} \neq \emptyset$ and $\max_{i\in\mathcal{P}} \omega(v_i) = 0$, any g-Elimination Set for G_{BFS} has to include $\{v_i | i \in \mathcal{P}\}$, so the conclusion also holds.

Now consider the case where $\max_{i\in\mathcal{P}} \omega(v_i) > 0$. Let T be a minimum-sized g-Elimination Set for G_{BFS}. T has to include at least $|\mathcal{P}| - 1$ nodes in $\{v_i | i \in \mathcal{P}\}$ because otherwise the un-included nodes in $\{v_i | i \in \mathcal{P}\}$ cannot be corrected (using the check node $\pi(v_{|V|+|C|})$). If $|T| = |\mathcal{P}| - 1$, the let $j \in \mathcal{P}$ be an integer such that $v_j \notin T$. If $T \cap \{v_i | i \in \mathcal{Q}\} = \emptyset$, 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 $\{v_i | i \in \mathcal{Q}\}$ will be corrected by iteration $\omega(v_j) - 1$, so this is an impossible case. So $T \cap \{v_i | i \in \mathcal{Q}\} \neq \emptyset$. 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 $\{v_i | i \in \mathcal{P}\} \subseteq T'$, 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 $\hat{G}_{sub}$.

Lemma 53. *Suppose $\max_{i\in\mathcal{P}} \omega(v_i) \leq \max_{i\in\mathcal{Q}} \delta(v_i)$. Consider five cases:*

1) *Case 1: If $|\mathcal{Q}| > 0$ and $\max_{i\in\mathcal{Q}} \delta(v_i) = k$, let S be a minimum-sized g-Elimination Set for G_{sub}.*
2) *Case 2: If $|\mathcal{Q}| > 0$, $\max_{i\in\mathcal{Q}} \delta(v_i) < k$ and $\delta(\pi(\pi(v_{|V|+|C|}))) \leq k$, let S be a minimum-sized g-Elimination Set for $\hat{G}_{sub}$ where $\delta(\pi(\pi(v_{|V|+|C|})))$ is changed to*

$$\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}} \delta(v_i) + 1\}.$$

3) *Case 3: If $|\mathcal{Q}| > 0$ and $\omega(\pi(\pi(v_{|V|+|C|}))) \leq \max_{i\in\mathcal{Q}} \delta(v_i) < k$, let S be a minimum-sized g-Elimination Set for G_{sub}.*
4) *Case 4: If $|\mathcal{Q}| > 0$ and $\max_{i\in\mathcal{Q}} \delta(v_i) < \omega(\pi(\pi(v_{|V|+|C|}))) \leq k$, let S be a minimum-sized g-Elimination Set for $\hat{G}_{sub}$ where $\delta(\pi(\pi(v_{|V|+|C|})))$ is changed to*

$$\max_{i\in\mathcal{Q}} \delta(v_i) + 1$$

and $\omega(\pi(\pi(v_{|V|+|C|})))$ is changed to

$$\infty.$$

5) *Case 5: If $|\mathcal{Q}| = 0$, there are two sub-cases: (1) if $\omega(\pi(\pi(v_{|V|+|C|}))) = 0$, let S be a minimum-sized g-Elimination Set for G_{sub}; (2) otherwise, let S be a minimum-sized g-Elimination Set for $\hat{G}_{sub}$ where $\delta(\pi(\pi(v_{|V|+|C|})))$ is changed to 1 and $\omega(\pi(\pi(v_{|V|+|C|})))$ is changed to ∞.*

Then $S \cup \{v_i | i \in \mathcal{P}\}$ is a minimum-sized g-Elimination Set for G_{BFS}.

Proof: By Lemma 52, there exists a minimum-sized g-Elimination Set for G_{BFS} that contains all the nodes in $\{v_i | i \in \mathcal{P}\}$. Now consider only minimum-sized g-Elimination Sets for G_{BFS} that contain all the nodes in $\{v_i | i \in \mathcal{P}\}$. See the nodes in $\{v_i | i \in \mathcal{P}\}$ 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 $P = \emptyset$, S is a minimum-sized g-Elimination Set for G_{BFS}* .

For Case 1, since $\max_{i\in\mathcal{Q}} \delta(v_i) = k$, the subtree rooted at $\pi(v_{|V|+|C|})$ cannot help correct the node $\pi(\pi(v_{|V|+|C|}))$ 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 $\delta(\pi(\pi(v_{|V|+|C|})))$. (For $\hat{G}_{sub}$, that value is $\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}} \delta(v_i) + 1\}$.) Now observe the check node $\pi(v_{|V|+|C|})$ and its neighboring variable nodes: when BP decoder runs on G_{BFS}, all the nodes in $\{v_i | i \in \mathcal{Q}\}$ can be corrected

automatically by iteration $\max_{i\in\mathcal{Q}}\delta(v_i) < k$; so by using the check node $\pi(v_{|V|+|C|})$, the node $\pi(\pi(v_{|V|+|C|}))$ can be corrected by iteration $\max_{i\in\mathcal{Q}}\delta(v_i)+1 \leq k$. That is equivalent to turning $\delta(\pi(\pi(v_{|V|+|C|})))$ into $\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}}\delta(v_i)+1\}$ and turning G_{sub} into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the assertion.

For Case 3, the node $\pi(\pi(v_{|V|+|C|}))$ needs to be corrected by iteration $\omega(\pi(\pi(v_{|V|+|C|})))$. But since the nodes in $\{v_i|i\in\mathcal{Q}\}$ will not all be corrected automatically until iteration $\max_{i\in\mathcal{Q}}\delta(v_i)$, and it takes one more iteration for the check node $\pi(v_{|V|+|C|})$ to propagate information to node $\pi(\pi(v_{|V|+|C|}))$, they cannot help decode $\pi(\pi(v_{|V|+|C|}))$. So for G_{sub}, it makes no difference whether the subtree rooted at $\pi(v_{|V|+|C|})$ is there or not when it comes to BP decoding. That leads to the assertion.

For Case 4, if we compare G_{sub} and $\hat{G}_{sub}$, we see that they differ only in their values of $\delta(\pi(\pi(v_{|V|+|C|})))$ and $\omega(\pi(\pi(v_{|V|+|C|})))$. Now observe the check node $\pi(v_{|V|+|C|})$ and its neighboring variable nodes: when BP decoder runs on G_{BFS}, all the nodes in $\{v_i|i\in\mathcal{Q}\}$ can be corrected automatically by iteration $\max_{i\in\mathcal{Q}}\delta(v_i)$; so the check node $\pi(v_{|V|+|C|})$ can help correct the node $\pi(\pi(v_{|V|+|C|}))$ by iteration $\max_{i\in\mathcal{Q}}\delta(v_i)+1 \leq \omega(\pi(\pi(v_{|V|+|C|})))$. That is equivalent to turning $\delta(\pi(\pi(v_{|V|+|C|})))$ into $\max_{i\in\mathcal{Q}}\delta(v_i)+1$, turning $\omega(\pi(\pi(v_{|V|+|C|})))$ to ∞ and turning G_{sub} into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the conclusion.

For Case 5, since $\mathcal{Q}=\emptyset$, the check node $\pi(v_{|V|+|C|})$ can help correct the node $\pi(\pi(v_{|V|+|C|}))$ in the 1st iteration. With an analysis similar to the above ones, we see that the assertion holds for both sub-cases. ■

Lemma 54. *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* $\omega(v_{i^*}) = \max_{i\in\mathcal{P}}\omega(v_i)$. *Consider two cases:*

1) *Case 1: If* $\max_{i\in\mathcal{P}}\omega(v_i) > \delta(\pi(\pi(v_{|V|+|C|})))$, *let* S *be any minimum-sized* g*-Elimination Set for* G_{sub}.
2) *Case 2: If* $\max_{i\in\mathcal{P}}\omega(v_i) \leq \delta(\pi(\pi(v_{|V|+|C|})))$, *let* S *be any minimum-sized* g*-Elimination Set for* $\hat{G}_{sub}$ *where* $\delta(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\infty$$

and $\omega(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\min\{\omega(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{P}}\omega(v_i)-1\}.$$

Then $S\cup\{v_i|i\in\mathcal{P}, i\neq i^*\}$ *is a minimum-sized* g*-Elimination Set for* G_{BFS}.

Proof: By Lemma 51, there exists a minimum-sized g-Elimination Set for G_{BFS} that includes the nodes in $\{v_i|i\in\mathcal{P}, i\neq i^*\}$ 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 $\pi(v_{|V|+|C|})$ cannot help correct the node $\pi(\pi(v_{|V|+|C|}))$. 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 $\omega(v)$). If $\pi(\pi(v_{|V|+|C|}))\in T^*$, the check node $\pi(v_{|V|+|C|})$ will correct v_{i^*} in the 1st iteration; otherwise, the BP decoder will correct $\pi(\pi(v_{|V|+|C|}))$ in at most $\delta(\pi(\pi(v_{|V|+|C|})))$ iterations, so $\pi(v_{|V|+|C|})$ will correct v_{i^*} in at most $\delta(\pi(\pi(v_{|V|+|C|})))+1 \leq \omega(v_{i^*})$ 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 $\pi(\pi(v_{|V|+|C|}))$ by iteration $\max_{i\in\mathcal{P}}\omega(v_i)-1 < \delta(\pi(\pi(v_{|V|+|C|})))$ because only then will the check node $\pi(v_{|V|+|C|})$ help correct the node v_{i^*} by iteration $\max_{i\in\mathcal{P}}\omega(v_i) = \omega(v_{i^*})$. That is equivalent to turning $\omega(\pi(\pi(v_{|V|+|C|})))$ into $\min\{\omega(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{P}}\omega(v_i)-1\}$, turning $\delta(\pi(\pi(v_{|V|+|C|})))$ into ∞ and turning G_{sub} into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the conclusion. ■

By using the above two lemmas repeatedly, we can reduce the $gSSE$ Problem from G_{BFS} to smaller and smaller subtrees, until the subtree contains only the root node v_1 (and v_1 will be included in the g-Elimination Set if and only if $\omega(v_1)\leq k$ at that moment). An algorithm based on the above idea is presented below.

Algorithm 55 *Exact Algorithm for* SSE_k

Input: Stopping Tree $G=(V\cup C, E)$, *integer* $k>0$.
Output: A k*-iteration Elimination Set of minimum size in* G.
Algorithm:

Generate G_{BFS} *by running BFS on* G.
for $i=1$ *to* $|V|+|C|$
{
  if $v_i\in V$
    $\delta(v_i)\leftarrow\infty, \omega(v_i)\leftarrow k$.
}
Let S *be an empty set.*
$i\leftarrow|V|+|C|$.
while $i\geq 1$
{
  if $i>1$ *and* $v_i\in V$
  {
    Let τ *be the minimum integer such that* v_τ *either is a sibling of* v_i *or is* v_i *itself.*
    $\mathcal{P}\leftarrow\{y|\tau\leq y\leq i, \omega(v_y)\leq k\}$.
    $\mathcal{Q}\leftarrow\{y|\tau\leq y\leq i, \delta(v_y)\leq k\}$.
    if $\max_{j\in\mathcal{P}}\omega(v_j)\leq\max_{j\in\mathcal{Q}}\delta(v_j)$
    {
      if $|\mathcal{Q}|>0, \max_{j\in\mathcal{Q}}\delta(v_j)<k$ *and* $\delta(\pi(\pi(v_{|V|+|C|})))\leq k$
      {
        $\delta(\pi(\pi(v_{|V|+|C|})))\leftarrow$
        $\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{j\in\mathcal{Q}}\delta(v_j)+1\}$.
      }
      else if $|\mathcal{Q}|>0$ *and* $\max_{j\in\mathcal{Q}}\delta(v_j)<$ $\omega(\pi(\pi(v_{|V|+|C|})))\leq k$

23) {
24) $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow \max_{j\in\mathcal{Q}} \delta(v_j) + 1$.
25) $\omega(\pi(\pi(v_{|V|+|C|}))) \leftarrow \infty$.
26) }
27) **else if** $|\mathcal{Q}| = 0$ *and* $\omega(\pi(\pi(v_{|V|+|C|}))) > 0$
28) {
29) $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow 1$.
30) $\omega(\pi(\pi(v_{|V|+|C|}))) \leftarrow \infty$.
31) }
32) $S \leftarrow S \cup \{v_j | j \in \mathcal{P}\}$
33) }
34) **else**
35) {
36) *Let* i^* *be an integer in* $\mathcal{P}$ *such that* $\omega(v_{i^*}) = \max_{j\in\mathcal{P}} \omega(v_j)$.
37) **if** $\max_{j\in\mathcal{P}} \omega(v_j) \leq \delta(\pi(\pi(v_{|V|+|C|})))$
38) {
39) $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow \infty$.
40) $\omega(\pi(\pi(v_{|V|+|C|}))) \leftarrow \min\{\omega(\pi(\pi(v_{|V|+|C|}))), \max_{j\in\mathcal{P}} \omega(v_j) - 1\}$.
41) }
42) $S \leftarrow S \cup \{v_j | j \in \mathcal{P}, j \neq i^*\}$.
43) }
44) $i \leftarrow \tau - 1$.
45) }
46) **else if** $i = 1$
47) {
48) **if** $\omega(v_1) \leq k$
49) $S \leftarrow S \cup \{v_1\}$.
50) $i = 0$.
51) }
52) **else**
53) $i \leftarrow i - 1$.
54) }
55) **Return** S.

Based on the previous analysis, we get the correctness of the algorithm.

Theorem 56. *Algorithm* 55 *returns an optimal (i.e., minimum-sized)* k*-iteration Elimination Set of* $G = (V \cup C, E)$.

The algorithm has time complexity $O(|V| + |C|)$.

VII. Conclusions

This paper studies the Stopping-Set Elimination Problem motivated by several applications, including the application of error correction based on natural redundancy and LDPC codes. The NP-hardness of both the SSE_∞ Problem and the SSE_1 Problem is proven. An approximation algorithm is presented for the SSE_1 Problem. And linear-time algorithms that return optimal solutions are presented for the SSE_∞ and SSE_k Problems when the Stopping Sets have tree structures.

References

[1] B. Addis, M. D. Summa and A. Grosso, "Removing Critical Nodes from a Graph: Complexity Results and Polynomial Algorithms for the Case of Bounded Treewidth," available at $http://www.optimization-online.org/DB_FILE/2011/07/3112.pdf$.

[2] L. Alvarez, P. Lions and J. Morel, "Image Selective Smoothing and Edge Detection by Nonlinear Diffusion. II," in *SIAM Journal on numerical analysis*, vol. 29, no. 3, pp. 845–866, 1992.

[3] R. Bauer and J. Hagenauer, "On Variable Length Codes for Iterative Source/Channel Decoding," in *Proceedings of Data Compression Conference*, pp. 273–282, 2001.

[4] A. Buades, B. Coll and J. Morel, "A Review of Image Denoising Algorithms, with a New One," in *Multiscale Modeling & Simulation*, vol. 4, no. 2, pp. 490530, 2005.

[5] P. Chatterjee and P. Milanfar, "Is denoising dead?" in *IEEE Transactions on Image Processing*, vol. 19, no. 4, pp. 895911, 2010.

[6] R. Coifman and D. Donoho, *Translation-invariant De-noising*, Springer, 1995.

[7] M. Fresia and G. Caire, "Combined Error Protection and Compression with Turbo Codes for Image Transmission Using a JPEG2000-like Architecture," in *Proc. ICIP*, pp. 821–824, 2006.

[8] T. Fujito, "Approximating Node-Deletion Problems for Matrodial Properties," in *Journal of Algorithms*, vol. 31, pp. 211–227, 1999.

[9] L. Guivarch, J. Carlach and P. Siohan, "Joint Source-channel Soft Decoding of Huffman Codes with Turbo-codes," in *Proceedings of Data Compression Conference (DCC)*, pp. 83-92, 2000.

[10] J. Hagenauer, "Source-controlled Channel Decoding," in *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2449–2457, 1995.

[11] M. Jeanne, J. Carlach and P. Siohan, "Joint Source-channel Decoding of Variable-length Codes for Convolutional Codes and Turbo Codes," in *IEEE Transactions on Communications*, vol. 53, no. 1, pp. 10-15, 2005.

[12] A. Jiang, Y. Li and J. Bruck, "Error Correction through Language Processing," in *Proc. IEEE Information Theory Workshop (ITW)*, 2015.

[13] A. Jiang, P. Upadhyaya, E. F. Haratsch and J. Bruck, "Error Correction by Natural Redundancy for Long Term Storage," in *Proc. Non-Volatile Memories Workshop (NVMW)*, 2017.

[14] A. N. Kim, S. Sesia, T. Ramstad, and G. Caire, "Combined Error Protection and Compression using Turbo Codes for Error Resilient Image Transmission," in *Proceedings of International Conference on Image Processing (ICIP)*, vol. 3, pp. III-912-15, 2005.

[15] J. Lansky, K. Chernik and Z. Vlckova. "Syllable-Based Burrows-Wheeler Transform," 2007.

[16] M. Kumar, S. Mishra, N. S. Devi and S. Saurabh, "Approximation Algorithms for Node Deletion Problems on Bipartite Graphs with Finite Forbidden Subgraph Characterization," in *Theoretical Computer Science*, vol. 526, pp. 90–96, 2014.

[17] Y. Li, Y. Wang, A. Jiang and J. Bruck, "Content-assisted File Decoding for Nonvolatile Memories," in *Proc. 46th Asilomar Conference on Signals, Systems and Computers*, pp. 937–941, Pacific Grove, CA, 2012.

[18] M. Lindenbaum, M. Fischer, and A. Bruckstein, "On Gabor's Contribution to Image Enhancement," in *Pattern Recognition*, vol. 27, no. 1, pp. 18, 1994.

[19] J. Luo, Q. Huang, S. Wang and Z. Wang, "Error Control Coding Combined with Content Recognition," in *Proc. 8th International Conference on Wireless Communications and Signal Processing*, pp. 1–5, 2016.

[20] G. Maral, M. Bousquet and Z. Sun, *Satellite Communications Systems: Systems, Techniques and Technology*, 5th edition, Wiley, 2010.

[21] E. Ordentlich, G. Seroussi, S. Verdu, and K. Viswanathan, "Universal Algorithms for Channel Decoding of Uncompressed Sources," *IEEE Trans. Information Theory*, vol. 54, no. 5, pp. 2243–2262, May 2008.

[22] E. Ordentlich, G. Seroussi, S. Verdu, M. Weinberger and T. Weissman, "A Discrete Universal Denoiser and Its Application to Binary Images," in *Proc. International Conference on Image Processing*, vol. 1, pp. 117, 2003.

[23] Z. Peng, Y. Huang and D. Costello, "Turbo codes for Image Transmission – A Joint Channel and Source Decoding Approach," in *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 18, no. 6, pp. 868–879, 2000.

[24] C. Poulliat, D. Declercq, C. Lamy-Bergot, and I. Fijalkow, "Analysis and Optimization of Irregular LDPC Codes for Joint Source-channel Decoding," *in IEEE Communications Letter*, vol. 9, no. 12, pp. 1064–1066, 2005.

[25] L. Pu, and Z. Wu, A. Bilgin, M. Marcellin, and B. Vasic, "LDPC-based Iterative Joint Source-channel Decoding for JPEG2000," in *IEEE Transactions on Image Processing*, vol. 16, no. 2, pp. 577–581, 2007.

[26] L. Rudin, S. Osher and E. Fatemi, "Nonlinear Total Variation based Noise Removal Algorithms," in *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259–268, 1992.

[27] T. J. Schaefer, "The Complexity of Satisfiability Problems," in *Proc. 10th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 216–226, 1978.

[28] C. E. Shannon, "Prediction and Entropy of Printed English," in *Bell System Technical Journal*, vol. 30, no. 1, pp. 50–64, 1951.

[29] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," in *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[30] M. Yannakakis, "Node-Deletion Problems on Bipartite Graphs," in *SIAM Journal on Computing*, vol. 10, no. 2, pp. 310–327, May 1981.

[31] Y. Wang, K. R. Narayanan and A. Jiang, "Exploiting Source Redundancy to Improve the Rate of Polar Codes," in *IEEE International Symposium on Information Theory (ISIT)*, Aachen, Germany, June 2017.

[32] Y. Wang, M. Qin, K. R. Narayanan, A. Jiang and Z. Bandic, "Joint Source-channel Decoding of Polar Codes for Language-based Sources," in *Proc. IEEE Global Communications Conference (Globecom)*, Washington D.C., December 2016.

[33] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu and M. Weinberger, "Universal Discrete Denoising: Known Channel," in *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 5–28, 2005.

[34] L. Yaroslavsky and M. Eden, *Fundamentals of Digital Optics: Digital Signal Processing in Optics and Holography*, Springer-Verlag New York, Inc., 1996.

.