Avoiding Trapping Sets in SC-LDPC Codes under Windowed Decoding

Allison Beemer
Department of Mathematics
University of Nebraska – Lincoln
Lincoln, Nebraska 68588
Email: allison.beemer@huskers.unl.edu

Christine A. Kelley
Department of Mathematics
University of Nebraska – Lincoln
Lincoln, Nebraska 68588
Email: ckelley2@math.unl.edu

***Abstract*—Spatially coupled low-density parity-check (SC-LDPC) codes are sparse graph codes that have recently become of interest due to their capacity-approaching performance on memoryless binary input channels. Moreover, protograph SC-LDPC codes have been shown to be asymptotically good under certain conditions. SC-LDPC codes are amenable to a windowed decoder that allows decoding of blocks of bits to be done serially, making them good candidates for applications such as streaming. Depending on how they are constructed, however, the codes may still be prone to error floors. One key element in protograph SC-LDPC code design is the edge-spreading method used to couple copies of the base protograph in the construction of the SC-protograph. In this paper, we present an edge-spreading algorithm that is designed to remove trapping sets from the resulting SC-protograph that are harmful with respect to the windowed decoder, and we examine how to determine which trapping sets should be removed in the construction.**

I. Introduction

In recent years, it was shown that *spatially coupling* several copies of a Tanner graph of an LDPC code improved their density evolution (DE) thresholds and brought them closer to channel capacity [1]. This phenomena, called *threshold saturation*, allows the SC-LDPC code to have the best threshold possible, i.e. the threshold under belief propagation (BP) using DE techniques approaches the maximum a posteriori (MAP) threshold. Further, it was shown that the threshold approaches capacity as the degree of the nodes in the graph, the spatial coupling length, and the memory or width of the coupling increase. While these results are all asymptotic, it is desirable for practical applications to design finite length SC-LDPC codes that have better performance both in the waterfall and in the error floor regions compared to standard LDPC codes of comparable code rates, block lengths, and node degrees.

Although there are several construction methods for SC-LDPC codes, one common method that is practical for implementation is based on protographs. First, a Tanner graph is chosen as a base protograph that is then replicated and *coupled* to form the *SC-protograph*. There are many ways to couple the edges from one copy to the other copies. This process of coupling is generally termed *edge-spreading*. The SC-LDPC code is then defined by a lift of the resulting SC-protograph.

The design of the SC-protograph is critical in terms of obtaining SC-LDPC codes with good thresholds and good error floor. While the threshold behavior is controlled mainly by the coupling width, coupling length, and the degree of the nodes in the SC-protograph, the error floor behavior is determined by the trapping sets in the SC-LDPC Tanner graph. The presence of trapping sets in the SC-LDPC graph is influenced by the trapping sets in the SC-protograph and the permutations used in lifting the edges of the SC-protograph (Note that a lift may maintain or improve the trapping set distribution, but will not make it worse). In contrast to other papers that optimize trapping sets for general protograph LDPC codes, we focus specifically on optimizing the trapping sets in the graph with respect to the windowed decoder [2] for SC-LDPC codes. We present an algorithm to remove harmful trapping sets during the edge-spreading step of the SC-LDPC construction in order to improve the resulting trapping set structure in the SC-protograph (with respect to the windowed decoder). We also show how trapping sets of a Tanner graph form a partially ordered set (poset) and explain how the poset can be used to identify which trapping sets should be prioritized for removal in the edge-spreading algorithm so that the most harmful are eliminated. While we focus mainly on trapping sets in this paper, the algorithm presented here may be adapted to absorbing sets and/or stopping sets as well. Indeed, several works address optimizing SC-LDPC codes with respect to absorbing sets [3], [4]; we note that many of these focus on array-based SC-LDPC codes.

This paper is organized as follows. In Section II, we provide some notation and background on SC-LDPC codes. In Section III, we present an edge-spreading algorithm that removes trapping sets according to some priority list. In Section IV, we explain how the poset structure of trapping sets may be used to prioritize the trapping sets to remove in the edge-spreading algorithm, and demonstrate with an example. We review two standard variations of the *windowed decoder* for SC-LDPC codes in Section V and discuss the relationship between trapping sets with respect to the windowed versus standard decoding. Section VI concludes the paper.

II. Preliminaries

In a protograph SC-LDPC construction, L copies of an LDPC Tanner graph, such as the one shown in Figure 1, are *coupled* to yield an SC-protograph. The coupling process may be thought of as first placing a copy of the base graph at

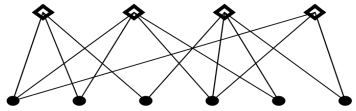

Fig. 1. Base Tanner graph to be coupled to form an SC-protograph. Variable nodes are denoted by •, and check nodes are denoted by ◇.

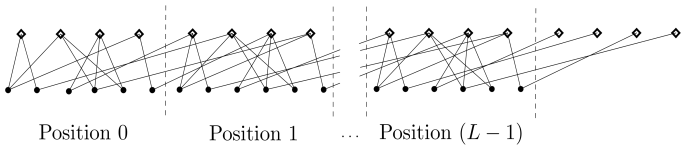

Fig. 2. SC-protograph resulting from randomly edge-spreading L copies of the Tanner graph in Figure 1 with coupling width $w = 1$, and applying the same map at each position.

positions $0, \ldots, L-1$, and then "edge-spreading" the edges in a special way to connect the variables nodes at position i to check nodes in copies $i, \ldots, w+i$ so that the degrees of the variable nodes in the base graph are preserved. The number L of positions is referred to as the *coupling length*, and the number w of future copies of the base graph that a variable node may spread to is called the *coupling width*. Terminating check nodes are introduced at the end of the SC-protograph as necessary to *terminate* the SC-protograph. An example of an SC-protograph obtained by coupling the LDPC Tanner graph in Figure 1 is given in Figure 2.

Edge-spreading is typically done in one of the following ways [1]: (1) For each variable node v in position $i \in \{0, \ldots, L-1\}$, if v has j neighbors $c_1, \ldots, c_j$ in the base Tanner graph, randomly choose for each $\ell = 1, \ldots, j$, a copy of c_ℓ from the positions $i, \ldots, w+i$. (2) if each variable node in position $i \in \{0, \ldots, L-1\}$ has j neighbors in the base Tanner graph, randomly choose j of the positions $i, \ldots, w+i$ to spread edges to, then, for each of the j neighbors $c_1, \ldots, c_j$ of a variable node v, randomly choose a check neighbor from the copies of c_ℓ ($\ell = 1, \ldots, j$) such that v has exactly one neighbor in each of the chosen j positions, and exactly one of each check node neighbor type. We will assume that the mapping given at Position 0 will be applied at all further positions. Indeed, doing so allows the resulting SC-LDPC code to be a terminated LDPC convolutional code if the permutations applied to lift the resulting SC-protograph are cyclic permutations [5], [6]. Terminated SC-LDPC convolutional codes are desirable for practical applications [3], [4], [5].

Without loss of generality, the coupling may be considered to be from left to right (from the variable node perspective), as suggested in the edge-spreading descriptions above. As a result of this structure, SC-LDPC codes are amenable to a *windowed* BP decoding process, where the BP decoding is performed on a window of variable nodes and check nodes and once these nodes are processed for some number of iterations, the window slides to the right and the nodes in the new window are processed. More detail is given in Section V.

A. Trapping Sets

Decoder failure of BP and other message passing decoders have been shown to be characterized by combinatorial structures in the graph, such as stopping sets, pseudocodewords, trapping sets, and absorbing sets. For many practical channels, the error floor behavior is dominated by the harmful (with respect to the decoder) trapping sets in the graph [7]. We now introduce relevant notation and terminology.

Following the notation of [8], suppose that the codeword $\mathbf{x}$ is transmitted, and $\hat{\mathbf{x}}$ is received. Let $\hat{\mathbf{x}}^l$ be the output after l iterations of the decoder run on the Tanner graph G.

Definition II.1. A node $\hat{\mathbf{x}}_i$ is said to be *eventually correct* if there exists $L \in \mathbb{N}$ such that $\hat{\mathbf{x}}_i^l = \mathbf{x}_i$ for all $l \geq L$.

Definition II.2. Let $\mathfrak{T}(\hat{\mathbf{x}})$ denote the set of variable nodes that are not eventually correct for a received word $\hat{\mathbf{x}}$, and let $G[\mathfrak{T}]$ denote the subgraph induced by $\mathfrak{T}(\hat{\mathbf{x}})$ and its neighbors in the graph G. If $G[\mathfrak{T}]$ has a variable nodes and b odd-degree check nodes, $\mathfrak{T}(\hat{\mathbf{x}})$ is said to be an (a, b)-*trapping set*. In this case, the set of variable nodes in error in the received word $\hat{\mathbf{x}}$ is called an *inducing set* for $\mathfrak{T}(\hat{\mathbf{x}})$.

Definition II.3. The *critical number* of a trapping set $\mathfrak{T}$ is the minimal number of variable nodes in an inducing set of $\mathfrak{T}$.

It should be noted that while the term "trapping set" refers to a set of variable nodes $\mathfrak{T}$, the subgraph $G[\mathfrak{T}]$ has a significant effect on the harmfulness of the trapping set during decoding. We will refer to $G[\mathfrak{T}]$ as a *trapping set subgraph*.

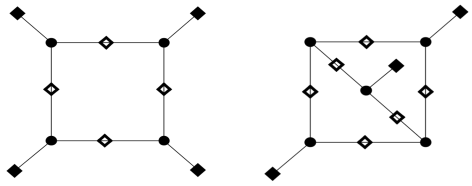

Fig. 3. Examples of a $(4,4)$ trapping set subgraph on the left, and a $(5,3)$ trapping set subgraph on the right. Variable nodes are denoted by •, even degree check nodes by ◇, and odd degree check nodes by ◆.

In [9], the authors present a trapping set ontology that gives a systematic way to determine which trapping sets are most relevant in characterizing decoder failure over the BSC. The authors assume Gallager A decoding, and use examples with column weight 3, but their work may be extended to other families. Figure 3 shows trapping set subgraphs under these assumptions. However, simulation results suggest that decoding failures on various channels with various decoders are topologically related, and that looking at trapping sets for the Gallager A/B algorithm on the BSC will give a good indication of how codes will perform with BP decoders [8].

In [10], the authors introduce the notion of *spread* to capture the harmfulness of *elementary* trapping sets, which are trapping sets whose check neighbors have degree one or two only. The idea of *spread* is that variable nodes connected to more degree-two than degree-one check nodes in the trapping set tend to propagate errors more.

Definition II.4. Let $\sigma_i(v)$ denote the number of check nodes incident to v that have i connections to trapping set $\mathfrak{T}$. The *spread* of a trapping set $\mathfrak{T}$ is the number of variable nodes $v \notin \mathfrak{T}$ for which $\sigma_2(v) > \sigma_1(v)$, with $\sigma_2(v) > 0$.

III. Avoiding trapping sets via edge-spreading

In this section we present an algorithm for edge-spreading that is designed to eliminate certain trapping set subgraphs in the resulting SC-protograph that are problematic with respect to the windowed decoder. We assume our base Tanner graph to be the Tanner graph of a block code, with no multi-edges and a reasonable block length. The algorithm assumes that a priority list of trapping sets to remove is provided at the start. Methods such as that presented in [11] may be used to identify trapping sets in the base graph. This list contains trapping sets from the base graph listed individually; that is, if two sets of variable nodes induce trapping set subgraphs that are isomorphic (and therefore, also of the same type), each will be listed in the priority list P as separate entries. In Section IV we describe how this list may be obtained using the trapping set poset structure; the list may also be obtained using methods from the trapping set ontology [9]. We adopt some terminology from the algorithm in [12] that is used to remove trapping sets in protograph LDPC codes.

We now describe how to spread edges from one position of variable nodes, given a coupling width of at most w.

Algorithm I

Let $P = \{\mathfrak{T}_1, \mathfrak{T}_2, \mathfrak{T}_3, \ldots, \mathfrak{T}_k\}$ be a list of trapping sets to avoid among variable nodes within each position of the SC-protograph, in order of priority. We will refer to P as the *priority list*.

1) Begin with L copies of the base Tanner graph, G, and initialize $S\,preadEdges = \emptyset$, $FrozenEdges = \emptyset$, and $S = \emptyset$. Set counter $i = 1$.
2) Choose $\mathfrak{T}_i$ from the priority list P. Let $E_{\mathfrak{T}_i}$ denote the set of edges in $G[\mathfrak{T}_i]$ at Position 0. If $E_{\mathfrak{T}_i} \cap S\,preadEdges \neq \emptyset$, go to Step 4. Otherwise, go to Step 3.
3) Choose an edge $e \in E_{\mathfrak{T}_i} \setminus FrozenEdges$ such that the incident check node, e_c, has degree strictly greater than 1 in $G[\mathfrak{T}_i]$. Spread this edge randomly to a copy of e_c in one of the available $w+1$ positions. Set $S\,preadEdges = S\,preadEdges \cup e$ and go to Step 4. If there is no such e, go to Step 5.
4) Let c be a check node (in Position 0) in $G[\mathfrak{T}_i]$ incident in $G[\mathfrak{T}_i]$ to an edge in $S\,preadEdges$. Set $FrozenEdges = FrozenEdges \cup E_c$, where E_c is the set of all edges incident to both $\mathfrak{T}_i$ and check type c. If $E_c \setminus S\,preadEdges \neq \emptyset$, set $S = S \cup \mathfrak{T}_i$. Go to Step 5.
5) Set $i := i + 1$. If $i \leq k$, go to Step 2. If $i = k + 1$, randomly spread each of the edges not in $FrozenEdges \cup S\,preadEdges$ to a copy of its incident check node in one of the $w + 1$ available positions, and stop.

To create the SC-protograph, the edge-spreading obtained by this algorithm at Position 0 is repeatedly applied at positions 1 through $L - 1$, where terminating check nodes are added as necessary.

The central idea of Algorithm I is to break apart the most harmful trapping set subgraphs appearing in the base graph by spreading their edges to later positions in the SC-protograph. By "freezing" edges which, within a trapping set subgraph on the priority list, share a check node with a spread edge, we ensure that we do not simply reconstruct the trapping set subgraph with check nodes in later positions. Because trapping set subgraphs contain cycles [11], choosing to spread an edge which is contained in a cycle, when possible, will have a greater chance of eliminating more trapping set subgraphs at once, and improving the girth of the SC-protograph. To avoid low-degree check nodes (e.g. degree 0 and 1) at either end of the SC-protograph, it may be necessary to adjust connections.

Lemma III.1. *The trapping sets in the set S at the termination of Algorithm I do not occur within a single position in the resulting SC-protograph. That is, if $\mathfrak{T} \in S$, then copies of the variable nodes in $\mathfrak{T}$ do not induce a subgraph isomorphic to $G[\mathfrak{T}]$ in any single position of the SC-protograph.*

Proof. Suppose that, after running Algorithm I, the trapping set $\mathfrak{T} = \{v_1, \ldots, v_a\} \in S$. Let $v_{j,i}$ denote the copy in Position j of the SC-protograph of variable node v_i in the base graph. Because $\mathfrak{T} \in S$, there exists some check node c in $G[\mathfrak{T}]$ which had at least one incident edge spread, and at least one incident edge frozen in Position 0. Thus, for all $j \in \{0, \ldots, L-1\}$, the subgraph of the SC-protograph induced by the variable nodes $\{v_{j,1}, \ldots, v_{j,a}\}$ contains at least two copies of check node c. Furthermore, this subgraph contains at least one copy of all other check nodes in $G[\mathfrak{T}]$. In particular, the subgraph induced by $\{v_{j,1}, \ldots, v_{j,a}\}$ has strictly more check nodes than $G[\mathfrak{T}]$, so the subgraphs cannot be isomorphic. □

Remark III.2. *Depending on the number of trapping sets in P it is possible that not all trapping sets in the priority list will end up in S; however, by ranking them in order of harmfulness (see Section IV), the most significant trapping sets will.*

IV. Ranking Trapping Sets

Algorithm I in Section III relies on a priority list that ranks the trapping sets to be removed in the base Tanner graph. Recall that Algorithm I cannot be iterated infinitely, as eventually the set $FrozenEdges$ will dominate the base graph. It is therefore important to construct the priority list judiciously. A ranking of the relative harmfulness of these trapping sets should take into account critical number and the number of small inducing sets, as well as the topological relations between trapping sets ([9], [8]). Thus, one option is to order all of the trapping sets of the base graph G by critical number. We note that, when possible, ties may be broken by the number of inducing sets of size equal to the critical number. That is, if $\mathfrak{T}_1$ and $\mathfrak{T}_2$ both have critical number $m(\mathfrak{T})$, but $\mathfrak{T}_1$ has more inducing sets of size $m(\mathfrak{T})$ than $\mathfrak{T}_2$, $\mathfrak{T}_1$ may be deemed more harmful than $\mathfrak{T}_2$. Constructing the priority list in this way forces Algorithm I to prioritize eliminating trapping sets from most harmful to least harmful. Such a ranking method was employed in [8] and [12].

A. Trapping Set Poset

A ranking based solely on critical number does not take the topological relations between trapping sets into account. A trapping set $\mathfrak{T}_1$ is a *parent* of the trapping set $\mathfrak{T}_2$ if $G[\mathfrak{T}_1]$ is a subgraph of $G[\mathfrak{T}_2]$. In this case, $\mathfrak{T}_2$ is a *child* of $\mathfrak{T}_1$, and in general $\mathfrak{T}_2$ is more harmful than $\mathfrak{T}_1$ [9]. Under the parent/child relationship, where $\mathfrak{T}_1 \leq \mathfrak{T}_2$ if and only if $\mathfrak{T}_1$ is a child of $\mathfrak{T}_2$, the set of trapping sets in a graph G forms a partially ordered set, or poset. Let $P(G)$ denote this poset.

Note that if a priority list arranged solely by critical number is used in Algorithm I, then edges of several child trapping set subgraphs of a parent trapping set may be frozen before arriving at that parent in the list. Doing so will freeze more edges early on, inhibiting progress through the list. Since avoiding a parent subgraph avoids its children, more trapping sets may be removed by simply reordering them. We now make this method precise.

Observe that eliminating a trapping set subgraph $G[\mathfrak{T}_i]$ in G eliminates the *down-set* generated by $\mathfrak{T}_i$,

$$D[\mathfrak{T}_i] = \{\mathfrak{T}_j \mid \mathfrak{T}_j \leq \mathfrak{T}_i\},$$

in $P(G)$. Following the notation of [13], we denote the poset of down-sets of $P(G)$ by $J(P(G))$. Note that $J(P(G))$ is a graded lattice, where the *rank* of an element of $J(P(G))$ is given by the size of the corresponding down-set in $P(G)$. If we wish to eliminate a set S of trapping sets in G deemed the most harmful, there is a unique minimal join of the down-sets of the elements of S, given by $D[S]$. Eliminating the maximal elements of this join down-set will eliminate the entire down-set, as will eliminating maximal elements of any down-set containing this join. Thus, we can label the elements of $J(P(G))$ according to how many maximal elements they contain (notice this is not necessarily order-preserving). To determine which parents to remove in order to remove a set S of trapping sets from G, we will look for the maximal element with minimal label in the up-set of $D[S]$, denoted $U(D[S])$, where $D[S]$ is an element of $J(P(G))$.

Consider the poset of trapping sets in Figure 4. Ordering by critical number gives the priority list $\{\mathfrak{T}_8, \mathfrak{T}_9, \mathfrak{T}_{10}, \mathfrak{T}_4, \mathfrak{T}_5, \mathfrak{T}_6, \mathfrak{T}_7, \mathfrak{T}_1, \mathfrak{T}_2, \mathfrak{T}_3\}$. However, if we are especially concerned with eliminating trapping sets with critical number 3 or smaller, it would be more efficient if we simply eliminated $\mathfrak{T}_5$ followed by $\mathfrak{T}_6$, or, even better, just $\mathfrak{T}_2$ (notice that all trapping sets of critical number at most 3 are contained in both $D[\mathfrak{T}_5, \mathfrak{T}_6]$ and $D[\mathfrak{T}_2]$). Similarly, to eliminate trapping sets with critical number at most 4, we could prioritize eliminating $\mathfrak{T}_1$ and then $\mathfrak{T}_2$ (since all trapping sets of critical number at most 4 are contained in $D[\mathfrak{T}_1, \mathfrak{T}_2]$).

Summarily, priority lists may be formed in many ways including by: (a) critical number, (b) spread, (c) taking the maximal elements in the poset, or (d) taking the maximal elements of a down-set containing the join of the down-sets.

B. Simulation results

Figure 5 shows the performance of SC-LDPC codes of block length 32000 bits and code rate 0.45 on the binary symmetric channel with Gallager A decoding. A base protograph

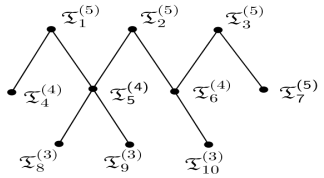

Fig. 4. Example of a poset of trapping sets of a given base Tanner graph, where $\mathfrak{T}_i^{(j)}$ denotes the ith trapping set, with critical number j.

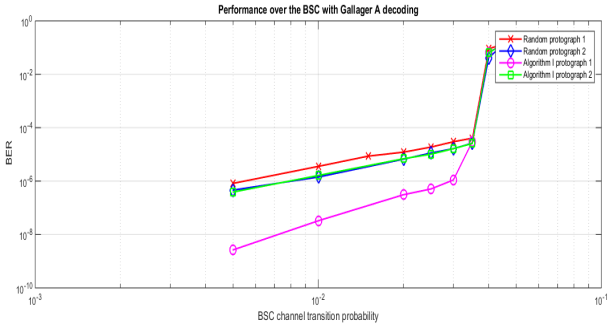

Fig. 5. A comparison of the performance of randomly-constructed protograph SC-LDPC codes to the performance of protograph SC-LDPC codes constructed using Algorithm I.

with 8 check and 16 variable nodes was coupled $L = 20$ times using different edge spreading algorithms. Random 1 and 2 were random edge spreading methods, whereas Algorithm I 1 and 2 were based on the proposed algorithm with priority lists ordered by eliminating parents first and then remaining trapping sets by critical number, and purely by critical number, respectively. The protographs were each lifted by a lifting factor $m = 100$ to yield SC-LDPC codes. The permutations for lifting the SC-protograph were chosen from the group of shifted identity matrices and were chosen randomly without any optimization. The figure shows Algorithm I significantly outperforming the other edge-spreading methods with more than two orders of improvement in the error floor region.

V. Windowed Decoder

Instead of decoding on the entire graph, Papaleo et al. and Iyengar et al. [2], [14] showed that the structure of SC-LDPC codes could be exploited to allow a *windowed decoder*, which could be viewed as moving along the graph from left to right. Recall that each *position* of variable nodes (resp., check nodes) in the SC-protograph contains the full set of variable nodes (resp., check nodes) in the base Tanner graph. There are two standard variations: either the window is defined to be W consecutive positions of variable nodes and all of their adjacent check nodes ([14]), which we will call a *$\mathcal{W}1$ window*, or is W consecutive positions of check nodes and all of their adjacent variable nodes ([2], [15]), which we will call a *$\mathcal{W}2$ window*.

In this paper, as in [2] and [15], we will assume that the window length W satisfies $w + 1 \leq W \leq L + w$. Recall that the SC-protograph is ultimately lifted to construct the SC-LDPC graph of the appropriate block length. Thus, the windowed

decoder will actually operate on the SC-LDPC graph and not on the SC-protograph. To say that the window contains variable nodes in certain positions of the SC-protograph means that it will contain the clouds of variable nodes located in those positions. Each decoding instance runs for a fixed number of iterations, seeking to decode the first position of variable nodes within the window that has not yet been decoded; the variable nodes in this position are called the *targeted symbols* [14], [15]. The set of edges contained in a window at a given decoding instance is the *window configuration* of the windowed decoder. Note that in windowed decoding, there will be variable node positions which have already been decoded for intermediate positions of the window frame. In particular, after the first wt iterations of a windowed decoder (assuming t iterations before each set of targeted symbols is released after the window slides), there will be w positions of already-decoded symbols preceding the current set of targeted symbols. These give an added degree of reliability to early check nodes within the window.

Whether the $\mathcal{W}1$ or $\mathcal{W}2$ window is used will change the window configuration, and hence the trapping sets to be considered within a window. Because trapping set subgraphs are induced by variable nodes, trapping set subgraphs in a $\mathcal{W}1$ window coincide with those in the entire SC-LDPC graph.

Lemma V.1. *If a subset $\mathfrak{T}$ of variable nodes within a $\mathcal{W}1$ window is a trapping set with respect to the windowed decoder, then $\mathfrak{T}$ is a trapping set of the same type with respect to the standard decoder on the SC-LDPC graph.*

Recall that a set X of variable nodes is an inducing set for a trapping set $\mathfrak{T}$ if when each node in X is in error from the channel, the variable nodes in $\mathfrak{T}$ are not eventually correct after any number of iterations. We now note that although trapping sets within a $\mathcal{W}1$ window are trapping sets in the SC-LDPC graph, the inducing sets may not coincide.

Lemma V.2. *If X is an inducing set of a trapping set $\mathfrak{T}$ in the SC-LDPC graph with respect to the standard decoder, then X may not be an inducing set of $\mathfrak{T}$ with respect to the $\mathcal{W}1$ windowed decoder.*

The relationship between trapping sets in a $\mathcal{W}2$ window and in the SC-LDPC graph is given next.

Lemma V.3. *If a subset $\mathfrak{T}$ of variable nodes within a $\mathcal{W}2$ window has all of its neighbors in the window and is a trapping set with respect to the windowed decoder, then $\mathfrak{T}$ is a trapping set of the same type with respect to the standard decoder on the SC-LDPC graph.*

Due to the constraints on window size, Lemma V.3 implies that any trapping set with respect to the $\mathcal{W}2$ windowed decoder contained entirely in the targeted symbols of a $\mathcal{W}2$ window is also a trapping set in the SC-LDPC graph with standard decoding.

We conclude by noting the effect of windowed decoding on the spread of (elementary) trapping sets. We define the *effective spread* of a trapping set $\mathfrak{T}$ contained entirely in a window of the windowed decoder to be the subset of the spread of $\mathfrak{T}$ in the SC-LDPC graph that is contained in the window. In a $\mathcal{W}1$ window, the effective spread may be a proper subset of the spread, while in the $\mathcal{W}2$ window, the spread and effective spread will be equal.

VI. Conclusion and Future Directions

We presented an algorithm that eliminates harmful trapping sets based on a predetermined priority list in the design of SC-protographs for SC-LDPC codes. The algorithm specifically targets the trapping sets with respect to the windowed decoder by avoiding them in sets of targeted variable nodes in the window. We showed how the poset structure for trapping sets may be used to construct the priority list. The methods presented in this paper may be combined with other methods that remove trapping sets at the lifting stage to yield stronger codes. More thorough simulations of SC-LDPC codes constructed using this algorithm with different priority lists and optimization at the lifting stage will appear in a forthcoming paper.

References

[1] S. Kudekar, T. Richardson, and R. L. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," *CoRR*, vol. abs/1201.2999, 2012. [Online]. Available: http://arxiv.org/abs/1201.2999

[2] M. Papaleo, A. R. Iyengar, P. H. Siegel, J. K. Wolf, and G. E. Corazza, "Windowed erasure decoding of ldpc convolutional codes," in *IEEE Info. Theory Workshop (ITW), Cairo*, Jan 2010, pp. 1–5.

[3] D. Mitchell, L. Dolecek, and J. Costello, D.J., "Absorbing set characterization of array-based spatially coupled ldpc codes," in *Proc. IEEE Int'l Symp. on Info. Theory (ISIT)*, June 2014, pp. 886–890.

[4] B. Amiri, A. Reisizadeh, J. Kliewer, and L. Dolecek, "Optimized array-based spatially-coupled ldpc codes: An absorbing set approach," in *Proc. IEEE Int'l Symp. on Info. Theory (ISIT)*, June 2015, pp. 51–55.

[5] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "Ldpc block and convolutional codes based on circulant matrices," *IEEE Trans. on Info. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec 2004.

[6] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, "Deriving good ldpc convolutional codes from ldpc block codes," *IEEE Trans. on Info. Theory*, vol. 57, no. 2, pp. 835–857, Feb 2011.

[7] T. Richardson, "Error-floors of ldpc codes," in *Proc. of 41st Allerton Conf. on Communication, Control and Computing*, 2003, pp. 1426–1435.

[8] D. V. Nguyen, S. K. Chilappagari, M. W. Marcellin, and B. Vasic, "On the construction of structured ldpc codes free of small trapping sets," *IEEE Trans. on Info. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr 2012.

[9] B. Vasic, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," in *Proc. of the 47th Allerton Conf. on Communication, Control, and Computing*, Sept 2009, pp. 1–7.

[10] S. Landner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured ldpc codes," in *Int'l Conf. on Wireless Networks, Comm. and Mobile Computing*, vol. 1, 2005, pp. 630–635.

[11] M. Karimi and A. H. Banihashemi, "An efficient algorithm for finding dominant trapping sets of irregular ldpc codes," in *Proc. IEEE Int'l Symp. on Info. Theory (ISIT)*, July 2011, pp. 1091–1095.

[12] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using tanner graph covers," *IEEE Trans. on Info. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug 2008.

[13] D. B. West, "Combinatorial mathematics," fall 2014 edition.

[14] A. R. Iyengar, P. H. Siegel, R. L. Urbanke, and J. K. Wolf, "Windowed decoding of spatially coupled codes," *IEEE Trans. on Info. Theory*, vol. 59, no. 4, pp. 2277–2292, Apr 2013.

[15] A. Iyengar, M. Papaleo, P. Siegel, J. Wolf, A. Vanelli-Coralli, and G. Corazza, "Windowed decoding of protograph-based ldpc convolutional codes over erasure channels," *IEEE Trans. on Info. Theory*, vol. 58, no. 4, pp. 2303–2320, Apr 2012.