Codes over Graphs

Lev Yohananov
Codes over Graphs

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Lev Yohananov

Submitted to the Senate of the Technion – Israel Institute of Technology
Adar 5782 Haifa February 2022
The research thesis was done under the supervision of Prof. Eitan Yaakobi in the Computer Science Department.

Acknowledgments

I would like to thank HaShem who sent me the right people, the ideas, the strength and the ability to write this thesis.

I am indebted to my advisor Prof. Eitan Yaakobi who guided the research and accompanied me closely with professionalism and humanity, and who has been the principal advisor since the beginning of my master studies.

Eitan, you strengthened my desire to investigate and research. Thank you for guiding me on how to arrange the ideas and define a problem. Thank you for leading me to independence in research. You threw me into the water and trusted me. You taught me to prepare a lecture and a presentation. You were available to help me anytime, anywhere. You generated a research group and created a wonderful atmosphere.

Thank you, Eitan, for your encouragement and leading me from the first step in the academic world to independence in research. Thank you for helping me receive scholarships and for caring about my success, even more than I cared for myself. Thank you for letting me learn from you in all areas, and I promise to make efforts to apply this knowledge.

Many thanks to the research group that is professional as well as fun. Thank you, the participants and organizers of the coding seminar, for a professional and supportive community.

I would like to thank all the wonderful friends I met in the faculty. You always showed interest, supported me when I was stuck and going through difficult moments, and shared my happiness with me in the most beautiful periods. You advised, directed, and painted my way in beautiful and pleasant colors.

In particular, I would like to thank the staff of the course "Combinatorics for Computer Science", which was my second home at the Technion, and headed by Prof. Eitan Yaakobi, Prof. Tuvi Etzion, Prof. Ronny Roth, Prof. Nader Bshouty and Prof. Shmuel Zaks. I am thankful to you for supporting and directing my personal development. You are a source of inspiration for me, of combining research and teaching. Thank you to all the staff, I have learned from each one of you so much regarding teaching and in general.

Thank you to the amazing and efficient secretarial staff, and to the great IT staff. You have always supported and taken an interest in my progress. You went far and beyond the requirements of your role.

Thank you to my family and friends who always supported and trusted me and who were genuinely interested and helped.

Special thanks to my dear parents, who pushed me to learn and study from a young age. Thank you for removing the obstacles from my path and for the countless hours and an enormous effort you invested for my studies and research. Thank you for always believing in me, and encouraging me to succeed. Thank
you for your endless love and dedication. I cannot express in a few lines, and in
even much more words, the gratitude I owe you.

Thank you to my faculty friends Yinon Horesh, Racheli Berman, Itay Fay-
erverker, Yehonatan Buchnik, Yakov Sokolik, Gil Kupfer, and Yuval Efron for
being with me all this time, and helped me not only in my research but also in
everyday life.

I am grateful to my special wife, who stood by me throughout the entire
period. Thank you for your support, encouragement, interest in my research,
and patience. You gave me the ability to work during the stressful periods, and
the strength to reach this moment. You were my pillar all along. Thank you to
the family of my wife for the support and help.

Thank you to my wonderful children. You gave me the strength and motiva-
tion. You taught me how to see things in proportion and many life skills. You
made me quicker and more efficient, doing tasks in parallel. You taught me many
things about myself that, without you, I would never have known.

This thesis is dedicated to all the dear people I care about. I would not have
made it this far without you. Thank you very much!

The generous financial support of the Technion, the Qualcomm and Jacobs
foundations and the Benin Scholarship Fund is gratefully acknowledged.
Publications List - Lev Yohananov

Journal Papers


Peer-Reviewed Conference Papers


3.7.2 Arbitrary $\rho$ ........................................ 53
3.8 Conclusion ........................................... 60
Appendix A .............................................. 60
Appendix B .............................................. 61
Appendix C .............................................. 63
Appendix D .............................................. 65
Appendix E .............................................. 66

4 Double and Triple Node-Erasure-Correcting Codes over Graphs 71
4.1 Introduction .......................................... 71
4.2 Definitions and Preliminaries ......................... 73
4.3 Optimal Binary Double-Node-Erasure-Correcting Codes .................................................................. 75
4.4 Single Node Regeneration ............................... 83
4.5 Binary Triple-Node-Erasure-Correcting Codes ............................................................................. 86
4.6 Decoding of the Triple-Node-Erasure-Correcting Codes ................................................................. 95
4.7 Conclusion .............................................. 96
Appendix A .............................................. 96
Appendix B .............................................. 101

5 Codes over Trees 109
5.1 Introduction .......................................... 109
5.2 Definitions and Preliminaries ......................... 112
5.3 Main Results ............................................ 116
5.4 Upper Bounds on Codes over Trees ................ 119
  5.4.1 Sphere-Packing Bound ............................ 120
  5.4.2 An Improved Upper Bound for $A(n,n-2)$ .......... 121
  5.4.3 An Improved Upper Bound for $A(n,n-3)$ .......... 122
5.5 Balls of Trees of Radius One ......................... 125
5.6 Balls of Trees of Arbitrary Radius .................. 129
  5.6.1 The Size of the Forest’s Ball ...................... 129
  5.6.2 The Size of the Tree Ball of Trees ................ 133
  5.6.3 The Average Ball Size .............................. 135
5.7 The Tree Balls of Trees for Stars and Path Trees ................................................................. 139
5.8 Constructions of Codes over Trees ................ 147
  5.8.1 A Construction of $T-(n, \lfloor n/2 \rfloor, n-1)$ Codes ................................................................. 147
  5.8.2 A Construction of $T-(n, n, n-2)$ Codes ................................................................. 149
  5.8.3 A Construction of $T-(n, \Omega(n^{n-2d}), d)$ Codes ..................................................... 149
  5.8.4 A Construction of $T-(n, \lfloor n/2 \rfloor, \lfloor \frac{n-1}{m} \rfloor, \lfloor \frac{2n}{3} \rfloor - \lfloor \frac{3n}{2m} \rfloor - 2)$ Codes ................................................................. 151
5.9 Conclusion .............................................. 156
Appendix A .............................................. 156
Appendix B .............................................. 157
6 Almost Optimal Construction of Functional Batch Codes

Using Hadamard Codes

6.1 Introduction

6.2 Definitions

6.3 A Construction of $FB-(2^s-1, s, \lfloor \frac{\alpha}{3} \cdot 2^{s-1} \rfloor)$ Codes

6.3.1 Graph Definitions

6.3.2 The $\text{FindGoodOrBadRequest}(G, t, B, v)$ function

6.3.3 The First Algorithm

6.3.4 The Second Algorithm

6.4 A Construction of $FB-(2^s + \lfloor (3\alpha - 2) \cdot 2^{s-2} \rfloor - 1, s, \lfloor \alpha \cdot 2^{s-1} \rfloor)$ Codes

6.5 A Construction of $FB-(2^{s+1} - 2, s, 2^s)$ Codes

6.6 A Construction of $FB-(2^s - 1, s, \lfloor \frac{\alpha}{3} \cdot 2^{s-1} \rfloor - s)$ Codes

6.6.1 The $\tau \leq n/4$ Case

6.6.2 The $\tau \leq n/3$ Case

6.6.3 Constructing Recovery Sets of Size 4

6.7 A Construction of $B-(2^s - 1, s, 2^{s-1})$ Codes

6.8 Conclusion

Appendix A

II Unpublished Published Journal Papers

7 Discussion

7.1 Codes over Graphs

7.2 Codes over Trees

7.3 Functional $k$-Batch Codes
List of Figures

3.1 The undirected graph $G$, its adjacency matrix $A_G$, and its lower-triangle-adjacency matrix $A'_G$ ................. 23
3.2 Two constructions of optimal binary single-node-erasure-correcting codes $SG$-[3,2] ........................................ 26
3.3 The neighborhood and the diagonal sets ..................................... 34
3.4 A decoding scheme of an optimal binary undirected double-node-erasure-correcting code $SU$-[11,9] ............... 42
3.5 The neighborhood and the diagonal sets ..................................... 43
3.6 A decoding scheme of an optimal binary undirected double-node-erasure-correcting code $SU$-[11,9] ............... 44
3.7 The neighborhood and the diagonal sets ..................................... 46
3.8 A decoding scheme of an optimal binary directed double-node-erasure-correcting code $SD$-[11,9] ............... 51
4.1 The constraints over undirected graphs, represented on the lower-triangle-labeling matrix .................................. 76
4.2 Demonstration for rebuilding ratio calculation for single node failure ................................................................. 84
4.3 The slope-two-diagonal constraints over undirected graphs, represented on the labeling matrix and the lower-triangle-labeling matrix ............................................................. 87
5.1 Presentation for a star and a path trees for $n = 5$ ....................... 114
5.2 An example of the $H_4$ set ....................................................... 123
5.3 The mapping $P_1$ .................................................................. 130
5.4 The idea of the recursive formula ................................................. 145
5.5 Hamiltonian path trees .............................................................. 148
5.6 A two-star tree ..................................................................... 152
6.1 The $G_x(G)$ graph. The green edges are the $x$-type edges and the dashed edges are the pair-type edges .................. 184
6.2 The graph $G_x(G')$ ................................................................ 185
6.3 Explanation of the function $FindGoodOrBadRequest(G, B, t, v)$ ................................................................. 189
List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Table of Definitions and Notations</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Table of Definitions and Notations</td>
<td>116</td>
</tr>
<tr>
<td>6.1</td>
<td>Table of Definitions and Notations</td>
<td>179</td>
</tr>
</tbody>
</table>
Abstract

In this work, codes over graphs are studied. A code over graphs is a set of complete graphs with self-loops with a fixed number of nodes n. Every graph-codeword in such codes stores the information symbols over some alphabet Σ on its edges. If a code over graphs contains only directed, undirected complete graphs with self-loops, then it is called a directed, undirected code over graphs, respectively. The number of edges in every codeword of the directed, undirected codes over graphs is \( n^2, \binom{n+1}{2} \), respectively. A node failure is the event in which the entire neighborhood set of edges of a node is erased. Codes that tolerate \( \rho \) node failures which are of maximal cardinality are called optimal. By using MDS codes, optimal codes over graphs are constructed over a field size \( \mathcal{O}(n^2) \). Several optimal constructions with smaller field sizes are presented in this work.

Another family of codes, called codes over trees, is studied. A code over trees is a set of undirected trees with a fixed number of nodes. Every tree-codeword corresponds to unique information which is given by the structure of the tree-codeword. These codes are constructed to tolerate any \( \rho \) edge erasures, which transform a tree into a forest with \( \rho + 1 \) sub-trees. The tree distance between two trees is the number of edges that belong to exactly one tree. Studying this distance measure is important for many applications that use trees and properties on their locality and the number of neighbor trees. The largest size of codes over trees with a prescribed minimum tree distance is investigated. In addition, upper bounds, several constructions, and decoding schemes are presented.

The last work of this thesis studies functional k-batch codes. A functional k-batch code of length n and dimension s encodes s information bits \( x_0, x_1, \ldots, x_{s-1} \) into n encoded bits \( y_0, y_1, \ldots, y_{n-1} \). These codes satisfy the following requirement: for any request of k linear combinations \( v_0, v_1, \ldots, v_{k-1} \) of the information bits, there are k pairwise disjoint subsets \( R_0, R_1, \ldots, R_{k-1} \subseteq \{0,1,\ldots,n-1\} \) such that \( \sum_{j \in R_i} y_j = v_i \). A code is called optimal if for given values of s and k it has the minimal number of the encoded bits. A recent conjecture states that for s information bits and any \( k = 2^{s-1} \) requests, the optimal solution requires \( 2^s - 1 \) encoded bits. This conjecture has been verified only for \( s \leq 5 \). Previous work showed that
codes with $n = 2^s - 1$ encoded bits can support $k = 2^{s-2} + 2^{s-4} + \left\lfloor \frac{2^{s/2}}{\sqrt{2}} \right\rfloor$ requests. This work reduces this gap and shows the existence of such codes for $k = \lfloor \frac{5}{6} 2^{s-1} \rfloor - s$ requests with the same number of encoded bits. Another construction in the work provides functional batch codes with $n = 2^{s+1} - 2$ servers and $k = 2^s$ requests, which are optimal codes.
Abbreviations and Notations

Abbreviations and Notations for Codes over Graphs

\( n \) — The number of nodes
\( G = (V_n, E) \) — An undirected complete graph over \( n \) nodes
\( A_G \) — The adjacency matrix of a graph \( G \)
\( L(v_i, v_j) \) — The label of the edge \( (v_i, v_j) \)
\( D-(n, M)_\Sigma, C_D \) — A code over directed graphs of length \( n \) and size \( M \)
\( U-(n, M)_\Sigma, C_U \) — A code over undirected graphs of length \( n \) and size \( M \)
\( D-[n, M]_\Sigma \) — A linear code over directed graphs of length \( n \) and size \( M \)
\( U-[n, M]_\Sigma \) — A linear code over undirected graphs of length \( n \) and size \( M \)
\( k_D \) — The dimension of a code over directed graphs
\( k_U \) — The dimension of a code over undirected graphs
\( R_D \) — The rate of a code over directed graphs
\( R_U \) — The rate of a code over undirected graphs
\( r_D \) — The redundancy of a code over directed graphs
\( r_U \) — The redundancy of a code over undirected graphs
\( SD-[n, k]_\Sigma \) — Systematic code over directed graphs
\( SU-[n, k]_\Sigma \) — Systematic code over undirected graphs
\( d(C_G) \) — The minimum distance of a code over graphs \( C_G \)
\( S_h \) — The neighborhood parity constraint
\( D_m \) — The diagonal parity constraint
Abbreviations and Notations for Codes over Trees

\begin{align*}
n & \quad \text{The number of nodes} \\
T(n) & \quad \text{The set of all labeled trees over } n \text{ nodes} \\
F(n, \delta) & \quad \text{The set of all forests with } \delta \text{ connected components} \\
F(n, \delta) & \quad \text{The size of } F(n, \delta) \\
d & \quad \text{The graph/tree distance} \\
T-(n, M, d) & \quad \text{A code over trees of size } M \\
A(n, d) & \quad \text{The largest size of a } T-(n, M, d) \text{ code} \\
r(n, d) & \quad \text{The minimum redundancy of a } T-(n, M, d) \text{ code} \\
\mathcal{H}_n & \quad \text{The set of forest-sets} \\
t & \quad \text{The radius of a ball} \\
B_T(n, t) & \quad \text{The tree ball of a tree of radius } t \text{ centered at } T \\
V_T(n, t) & \quad \text{The size of } B_T(n, t) \\
V(n, t) & \quad \text{The average ball size of radius } t \\
V^*(n, t)) & \quad \text{The value of } V_T(n, t) \text{ if } T \text{ is a star} \\
V^-(n, t) & \quad \text{The value of } V_T(n, t) \text{ if } T \text{ is a path tree} \\
S_T(n, t) & \quad \text{The sphere of a tree of radius } t \text{ centered at } T \\
S_T(n, t) & \quad \text{The size of } S_T(n, t) \\
S^+(n, t)) & \quad \text{The value of } S_T(n, t) \text{ if } T \text{ is a star} \\
S^-(n, t) & \quad \text{The value of } S_T(n, t) \text{ if } T \text{ is a path tree} \\
P_T(n, t) & \quad \text{The forest ball of a tree of radius } t \text{ centered at } T \\
P_T(n, t) & \quad \text{The set of profiles of } P_T(n, t) \\
B_F(n, t) & \quad \text{The tree ball of a forest of radius } t \text{ centered at } F \\
V_F(n, t) & \quad \text{The size of } B_F(n, t)
\end{align*}
Abbreviations and Notations for Functional $k$-Batch Codes

$FB-(n,s,k)$ — A func. $k$-batch code of length $n$ and dimension $s$

$R_i$ — The $i$-th recovery set

$(G,B,R)$ — A triple-set

$\mathcal{M}(G,B,R)$ — A triple-matrix of $M$

$e$ — A unit vector of length $s$ with 1 at its last index

$M$ — A request matrix

$v_i, w_i$ — The $i$-th request/column vector in $M, \mathcal{M}$

$G$ — An $HG$-matrix

$g_i$ — A column vector in $G$ representing the $i$-th server

$G_x(G)$ — An $x$-type graph of $G$

$C_x(G)$ — The partition of simple cycles of $G_x(G)$

$P_x(g_i, g_j)$ — A simple path between $g_i$ and $g_j$ in $G_x(G)$

$d_{P_x}(g_i, g_m)$ — The sub-length from $g_i$ to $g_m$ in $P_x(g_i, g_j)$

$F_x(g_i, g_j)$ — A reordering function for a good-path $P_x(g_i, g_j)$
A Note to the Reader

This dissertation is based on the following publications.

Journal Papers


Peer-Reviewed Conference Papers


All these research works were done in collaboration with my supervisor, Prof. Eitan Yaakobi. He introduced me with the field, referred me to the relevant literature, and helped with asking the right questions. He suggested methods for solving the problems, and of course accompanied the research along all the way, including the writing of the above papers. The paper [J2] is an extension work of [J1] in which we collaborated with Yuval Efron.

This dissertation contains three main parts. The first part contains the published journal papers [J1] and [J2], one chapter for each paper. The second part is dedicated to the published journal paper [J3]. Both parts are dealing with codes over graphs. The third part is dedicated to the published conference paper [C5] and the submitted unpublished journal paper [J4]. This part is dealing with functional $k$-batch codes.

Contribution of the authors: All the works are related to the main research topic of this work - tolerating erasures of edges on directed, undirected graphs and trees. The results of papers [J1] and [J3] were derived by collaboration with my supervisor, Prof. Eitan Yaakobi. Some of results in [J2] were derived by collaboration with Yuval Efron. This paper is a direct extension work of [J1]. Yuval’s contribution is reflected in the results of Chapters 3 and 4. His contribution is as follows: Chapter 3 was written in a half partnership between me and Yuval. This chapter was derived directly from the result of our first work [J1]. Chapter 4 was written by Yuval under the guidance of Eitan Yaakobi. The result of Chapter 4 is an interesting result, but it is not the main result of this work. Chapter 5 was derived and written by me under the guidance of Eitan Yaakobi. This result is the main result of this work.

Following the instructions of the Technion’s Graduate School, the structural of this dissertation is as follows. The first chapter, Introduction, presents a comprehensive and up-to-date overview of all areas of research. Afterwards, the Research Methods chapter describes the techniques developed and used during the work. Then, we present the conference and journal papers organized in the two parts as described above, where each paper contains its own list of references. The last chapter, Discussion, includes a brief summary of the results while taking into account the coherence and integration of the entire work. The separate bibliography which is given in p. 225 lists the references which are given in the Introduction, the Research methods, and the Discussion chapters.
Chapter 1

Introduction

There are many different types of codes that have been studied in the past. While classical codes and array codes were studied in-depth, only little is known about codes that store the information in different forms. In this work, we study codes that store the information in a graph. The motivation of this method can be used in several information systems like neural networks, associative memories, and distributed systems. The first system consists of the neural units that are connected via links which store and transmit information between the neural units [12]. The second system stores the information by associations between different data items [32]. Similarly, the last one stores the information between nodes [7]. In this work, we establish two families of codes which are called codes over graphs and codes over trees.

A code over graphs is a set of complete graphs with self-loops and a fixed number of nodes $n$. The information symbols over an alphabet $\Sigma$ are stored on the edges of every graph-codeword. A family of codes over graphs that contains only directed complete graphs with self-loops is called a code over directed graphs. Similarly, if it contains only undirected complete graphs with self-loops then it is called a code over undirected graphs. For directed codes over graphs, the number of symbols will be $n^2$ and for undirected codes over graphs, it will be $\binom{n+1}{2}$. A node failure is the event in which the entire neighborhood set of edges of a node is erased. The minimum number of redundant edges of any code over directed, undirected graphs of length $n$, is $2n\rho - \rho^2, n\rho - \binom{\rho}{2}$, respectively. Codes with the minimal number of redundancy edges that tolerate $\rho$ node failures are called optimal. While the construction of optimal codes over graphs can be easily accomplished by MDS codes, their field size has to be $O(n^2)$, when $n$ is the number of nodes in the graph. By using product codes, the field size is reduced to $O(n)$, and by rank-metric codes, binary close to optimal constructions are presented. Furthermore, for prime $n$, binary codes over graphs that correct two and three-node failures are presented in this work. While the former construction is optimal, the latter is within one bit away from optimality. Finally, for a constant $k$, upper bounds on the number of nodes for optimal codes over graphs that tolerate $\rho = n - k$ node failures, are presented.
A code over trees is a set of undirected trees with a fixed number of nodes $n$. In this work, we only study undirected trees, while the directed case is left for future work. In this model, the information is the unique structure of the tree-codewords. For example, the star tree is a tree that has exactly one node which is not a leaf. The path tree is a tree that has exactly two leaves and all the other nodes are of degree 2. Practically, the information stored by such codes depends on the application that will be used. For example, in binary search trees, the information values and pointers that represent edges are stored in nodes, and in tries or suffix trees, the symbols or strings are stored on edges. An edge erasure is the event in which one of the edges in the tree is erased and a forest is received with two connected components. This is also extended to the erasure of multiple edges. If $t$ edges are erased, then a forest with $t + 1$ connected components is received and the number of such forests is $\binom{n-1}{t}$. The task of codes over trees is to correct such edge erasures and to reconstruct the original tree. In other words, a code over trees that corrects any $\rho$ edge erasures is able to reconstruct the original tree from a received forest that has at most $\rho + 1$ connected components.

In this work, several constructions on codes over trees are presented. In fact, the correction ability of codes over trees can be achieved by defining the so-called tree distance metric. The tree distance between two trees is defined to be the number of edges that belong to exactly one tree. The largest size of codes over trees with a prescribed minimum tree distance is investigated. The radius-$t$ ball of a tree $T$ is the set of all trees whose tree distance from $T$ is at most $r$. A significant part of this study is dedicated to the problem of calculating the size of such balls of a given radius. These balls are not regular and we show that while star trees have asymptotically the smallest size of the ball, the maximum is achieved for path trees.

This work also deals with another family of codes called functional $k$-batch codes. A functional $k$-batch code of length $n$ and dimension $s$ consists of $n$ encoded bits $y_0, y_1, \ldots, y_{n-1}$ of $s$ information bits $x_0, x_1, \ldots, x_{s-1}$. For all $0 \leq j \leq n-1$ there exists $0 \leq \ell_j \leq s-1$ such that $y_j = x_{i_0} + x_{i_1} + \cdots + x_{i_{\ell_j}-1}$, for some distinct integers $0 \leq i_0, i_1, \ldots, i_{\ell_j-1} \leq s-1$. These codes satisfy the following requirement. For any request of $k$ linear combinations $v_0, v_1, \ldots, v_{k-1}$ (not necessarily distinct) of the information bits, there are $k$ pairwise disjoint subsets $R_0, R_1, \ldots, R_{k-1} \subseteq \{0, 1, \ldots, n-1\}$ such that the sum of the linear combinations in the related encoded bits of $R_i$, $0 \leq i \leq k-1$, is $v_i$, i.e., $\sum_{j \in R_i} y_j = v_i$. In other words, a set of encoded bits can be used together for recovering a single request, and it cannot be used for other requests. The goal under this paradigm is to find the minimum number of encoded bits for given values of $s$ and $k$, and in that case, the code is called optimal. An open conjecture states that for $k = 2^s-1$ requests, it is enough to have $n = 2^s - 1$ encoded bits. Indeed, this conjecture was proven to be correct for $s \leq 5$. However, the general case is still unknown. In this work we significantly improve the known results, by showing that $2^s - 1$ encoded bits can recover any $k = \lceil \frac{5}{6}2^s-1 \rceil - s$ requests. In addition, several interesting new constructions are presented.
1.1 Codes over Graphs

A code over graphs is a set of complete graphs over \( n \) nodes, with self-loops. Such a set can consist either only from undirected graphs, called code over undirected graphs, or only from directed graphs, called code over directed graphs. In both cases, it stores the information on graph-codeword edges, i.e., symbols over \( \Sigma \). Such codes are designed for tolerating node failures, which are the erasures of whole neighborhoods of the nodes. By definition of graph-codewords, a code over graphs can be represented as a set of square \( n \times n \) matrices over \( \Sigma \), called an adjacency matrix. In this representation, a failure of node \( i \) of fixed graph-codeword is equivalent to the failure, i.e., erasure, of row \( i \) and column \( i \) in its adjacency matrix. Therefore a code over graphs is a variation of array code with the above specific pattern of failure. In this work, we seek to construct such codes with as small as possible alphabet size, while tolerating as many node failures as possible.

Since a code over graphs is a variation of an array code, we use some ideas from a list of related work, i.e., the maximum-rank array codes [25], d-codes [28], B codes [21], EVENODD codes [22], STAR codes [10], RDP code [18], X-codes [19], and regenerating codes [20, 22, 24]. Note, that the failure pattern of a code over graphs is different from these array codes. Moreover, an undirected code over graphs is a set of square symmetric matrices, while array codes are not necessarily symmetric or square. Therefore, the most relevant work for undirected case is \( d \)-codes [28] of Schmidt.

Mathematically speaking, a code over directed graphs is called a directed \( \rho \)-node-erasure-correcting code if it can correct the failure of any \( \rho \) nodes in each graph in the directed code. An undirected \( \rho \)-node-erasure-correcting code is defined similarly. The minimum redundancy \( r_D \), \( r_U \) of any directed, undirected \( \rho \)-node-erasure-correcting code of length \( n \), satisfies

\[
\begin{align*}
    r_D &\geq n^2 - (n - \rho)^2 = 2n\rho - \rho^2, \\
    r_U &\geq \binom{n + 1}{2} - \binom{n - \rho + 1}{2} = n\rho - \binom{\rho}{2},
\end{align*}
\]

respectively. A code over directed, undirected graphs satisfying the first, second inequality with equality will be called optimal, respectively. Note that optimal codes can be represented systematically by placing the information on the edges of the first \( k \) nodes. Hence, for systematic codes over graphs, the number of redundancy nodes is at least \( \rho \). Note that for all \( n \) and \( \rho \), one can always construct an optimal directed \( \rho \)-node-erasure-correcting code from an \( [n^2, (n - \rho)^2, 2n\rho - \rho^2 + 1] \) MDS code. Similarly, one can always construct an optimal undirected \( \rho \)-node-erasure-correcting code from an \( [\binom{n+1}{2}, \binom{n-\rho+1}{2}, n\rho - \binom{\rho}{2} + 1] \) MDS code. However, in both cases, the field size of the graph codes will be at least \( \Theta(n^2) \).

In this thesis we improve these straight forward constructions and also present upper bounds on the size of both directed and undirected codes over graphs.
We want to note that finding optimal and binary codes over graphs, is one of the most difficult tasks in this work. In this work we present two interesting binary constructions. The first is a construction for codes tolerating double node failures for both directed and undirected cases, when the number of nodes is a prime number, together with two efficient encoding and decoding algorithms for both constructions. By defining a minimal graph distance for these codes (which is defined by vertex cover metric), we proved that they are of minimum graph distance 3. Therefore, these codes are capable of correcting a single error of a node. The second is a construction for triple-node-erasure-correcting codes when the number of nodes is a prime number and two is a primitive element in $\mathbb{Z}_n$. These codes are at most a single bit away from optimality. Finding an optimal construction for this case is left for future work. It is also interesting to find a construction that tolerates three node failures for the directed case.

Another result followed the recent works on regenerating codes [7, 24, 29] in order to analyze a sufficient number of edges we have to read in order to correct a single node erasure while using our double-node-erasure-correcting code. We will show that in order to correct a single node erasure, we are not required to read the rest of the graph in its entirety. Namely, while the number of edges in the graph is $\frac{n(n+1)}{2}$, we will show that it is enough to read only $\frac{5}{12} n^2 + O(n)$ edges in order to decode a single node failure.

1.2 Codes over Trees

In this chapter, we define a new family of codes which will be called codes over trees. Each codeword-tree in such a code will be a tree with a fixed number of nodes. Every codeword-tree corresponds to unique information that is stored, sent, or read, i.e., the information is the structure of the codeword-tree. The storage of information depends mainly on the application that will be used. For example, in binary search trees [5] the information values and pointers that represent edges are stored in nodes. In tries or suffix trees [14] the symbols or strings are stored on edges. Such codes will be constructed with the ability to reconstruct trees with erroneous or erased edges. In order to deal with erasures and errors of edges of trees, we initiate the study of codes over trees. In this work, we found several constructions and bounds on the size of such codes.

Motivated by the coding theory approach, in this work we apply the tree distance, which is a metric, to study codes over trees with a prescribed minimum tree distance. This family of codes can be used for the correction of edge erasures. There are several applications in which such codes can be used. For example, in data structures, a tree is a widely used abstract data type that simulates a hierarchical tree structure [5]. Such tree data structures store the information in nodes and use edges as pointers between them. There are numerous examples for such tree data structures including abstract syntax trees (AST), parsing trees and binary search trees (BST) [5, 14]. AST represent the abstract syntactic structure...
of source code written in a programming language, while each node of the tree
denotes a construct occurring in the source code. Parsing trees represent the
syntactic structure of a string according to some context-free grammar. BST
trees store in each node a value greater than all the values in the node’s left
subtree and less than those in its right subtree. These tree data structures can
be implemented such that each node stores a list of pointers to other nodes in the
tree. Theoretically, such pointers might have wrong addresses, which affects the
reliability of the data structure. By adding redundancy edges and nodes, codes
over trees may correct the unexpected pointer mismatches. Another family of
applications include data structures such as tries and suffix trees [14] in which the
information is stored on the edges rather than the nodes. Such data structures
can be implemented by a list of $n - 1$ edges which is a list of node pairs together
with the information on every edge. Again, theoretically, such an edge list may
have failures that can indeed be corrected using classical error-correction codes.
However, these codes will not be cardinality optimal since they do not take
advantage of the structure of the tree. For the binary case, using classical error-
correction codes, we show in this work the construction of codes over trees of
size $\Omega(n^{n-2d})$ where $d \leq n/2$ corresponds to the minimum tree distance of the
code. Using codes over trees we show that it is possible to construct codes of
cardinality $\Omega(n^2)$, while the minimum tree distance $d$ approaches $\lfloor 3n/4 \rfloor$ and $n$
is a prime number.

Another direction of this study is to define and study the size of balls accord-
ing to the so-called tree distance. Namely, given two labeled trees over $n$ nodes,
the tree distance is defined to be half of the minimum number of edges that are
required to be removed and added in order to change one tree to another. This
value is also equivalent to the difference between $n - 1$ and the number of edges
that the two trees share in common. Despite the popularity of this distance func-
tion, the knowledge of its characteristics and properties is quite limited. The goal
of this work is to close on these gaps and study trees under the tree distance from
a coding theory perspective. This investigation is useful not only for applying
the sphere packing bound on codes over trees, but also for other applications
such as computer vision and pattern recognition. One of the classical problems
in graph theory is finding a minimum spanning tree (MST) for a given graph.
While the MST problem is solved in polynomial time [17, 20], it may become
NP-hard under some specific constraints. For example, in the degree-constrained
MST problem ($d$-MST) [16, 21, 22, 35], it is required that the degree of every
vertex in the MST is not greater than some fixed value $d$. In another example,
the goal is to look for an MST in which the length of the simple path between
every two vertices is bounded from above by a given value $D \geq 4$ [23]. One of the
common approaches for solving such problems uses evolution algorithms (EA).
Under this setup, the goal is to find a feasible tree to the problem by iteratively
searching for a candidate tree. This iterative procedure is invoked by using mu-
tation operations over the current tree in order to produce a new candidate tree.
These mutation operations typically involve the modification of edges in the tree.
and as such are highly related to the tree distance. Thus, in order to analyze the complexity of such algorithms, it is necessary to study the size of the balls according to the tree distance. In fact, in [11] the size of the radius-one ball was computed for all trees with at most 20 vertices. According to this computer search, it was observed that the smallest size of the ball is achieved when the tree is a star tree (i.e., the tree has one node connected to all other nodes), while the largest for a path tree (i.e., the tree has two leaves and the degree of all other nodes is two). In this work, we establish this result for any number of nodes in the tree as well as for any radius. Furthermore, it is shown that the size of the radius-$t$ ball ranges between $\Omega(n^{2t})$ (for a star tree) and $O(n^{2t})$ (for a path tree), while the average size of all balls is $\Theta(n^{2.5t})$.

1.3 Functional $k$-Batch Codes

Batch codes were first proposed by Ishai et al. [13]. These codes were motivated by several applications for load-balancing in storage and cryptographic protocols. A batch code encodes a length-$s$ string $x$ into $n$ strings, where each string corresponds to a server, such that each batch request of $k$ different bits (and more generally symbols) from $x$ can be decoded by reading at most $t$ bits from every server. This decoding process corresponds to the case of a single-user. There is an extended variant for batch codes [13], which is intended for a multi-user application instead of a single-user setting, known as multiset batch codes. Such codes have $k$ different users and each user requests a single data item. Thus, the $k$ requests can be represented as a multiset of the bits since the requests of different users may be the same, and each server can be accessed by at most one user.

Mathematically speaking, an $FB-(n,s,k)$ functional $k$-batch code (and in short $FB-(n,s,k)$ code) of dimension $s$ consists of $n$ servers storing linear combinations of $s$ linearly independent information bits. Any multiset of size $k$ of linear combinations from the linearly independent information bits, can be recovered by $k$ disjoint subsets of servers. If all the $k$ linear combinations are the same, then the servers form an $FP-(n,s,k)$ functional $k$-Private Information Retrieval (PIR) code (and in short $FP-(n,s,k)$ code). Clearly, an $FP-(n,s,k)$ code is a special case of an $FB-(n,s,k)$ code. It was shown that functional $k$-batch codes are equivalent to the so-called linear parallel random I/O (RIO) codes, where RIO codes were introduced by Sharon and Alrod [26], and their parallel variation was studied in [27, 30]. Therefore, all the results for functional $k$-batch codes of this paper hold also for parallel RIO codes. If all the $k$ linear combinations are of a single information bit (rather than linear combinations of information bits), then the servers form an $B-(n,s,k)$ $k$-batch code (and in short $B-(n,s,k)$ code).

The value $FP(s,k), B(s,k), FB(s,k)$ is defined to be the minimum number of servers required for the existence of an $FP-(n,s,k), B-(n,s,k), FB-(n,s,k)$ code, respectively. Several upper and lower bounds can be found in [34] on these values.
Wang et al. [31] showed that for $k = 2^{s-1}$, the length of an optimal $k$-batch code is $2^s - 1$, that is, $B(s, k = 2^{s-1}) = 2^s - 1$. They also showed a recursive decoding algorithm. It was conjectured in [31] that for the same value of $k$, the length of an optimal functional batch code is $2^s - 1$, that is, $FB(s, k = 2^{s-1}) = 2^s - 1$. Indeed, in [33] this conjecture was proven for $s = 3, 4$, and in [34], by using a computer search, it was verified also for $s = 5$. However, the best-known result for $s > 5$ only provides a construction of $FB-(2^s - 1, s, 2^{s-2} + 2^{s-4} + \left\lceil \frac{2^{s/2}}{\sqrt{2^4}} \right\rceil)$ codes [34]. This paper significantly improves this result and reduces the gap between the conjecture statement and the best-known construction. In particular, a construction of $FB-(2^s - 1, s, \left\lceil \frac{5}{6} \cdot 2^{s-1} \right\rceil - s)$ codes is given. To obtain this important result, we first show an existence of $FB-(2^s - 1, s, \left\lceil \frac{3}{4} \cdot 2^{s-1} \right\rceil)$ code. Moreover, we show how to construct $FB-(2^s + \left\lceil (3\alpha - 2) \cdot 2^{s-2} \right\rceil - 1, s, \left\lceil \alpha \cdot 2^{s-1} \right\rceil)$ codes for all $2/3 \leq \alpha \leq 1$. Another result that can be found in [34] states that $FP(s, 2^s) \leq 2^{s+1} - 2$. In this case, the lower bound is the same, i.e., this result is optimal, see [10]. In this paper we will show that this optimality holds not only for functional PIR codes but also for the more challenging case of functional batch codes, that is, $FB(s, 2^s) = 2^{s+1} - 2$. Lastly, we show a non-recursive decoding algorithm for $B-(2^s, s, k = 2^{s-1})$ codes. In fact, this construction holds not only for $k$ single bit requests (with respect to $k$-batch codes) but also for $k$ linear combinations of requests under some constraint that will be explained in the paper. All the results in the paper are achieved using a generator matrix $G$ of a Hadamard codes [2] of length $2^s$ and dimension $s$, where the matrix’s columns correspond to the servers of the $FB-(n, s, k)$ code.
Chapter 2

Research Methods

This chapter provides an overview of several techniques which were used in the research described in this dissertation. It includes many mathematical methods which are derived from combinatorics, discrete mathematics, and information theory. We also used computer search algorithms for several purposes.

In papers [J1], [J2] we provide several constructions and upper bounds for codes over graphs. We apply some ideas from [3] obtaining the main construction of undirected codes over graphs in [J1]. The directed case is a non-trivial extension of the undirected case; see [J1]. We showed a decoding algorithm for both constructions and used the mathematical induction proving the correctness of these algorithms. The upper bounds are derived using combinatorial, algebraic, and information theory tools.

We applied the idea of the construction of Maximum-rank array codes for square matrices in [25] and obtained the construction of triple-node-erasure-correcting code [J2]. This result was found using computer search algorithm. We proved this result afterwards involving several techniques from discrete mathematics and ring theory.

This is the first work which introduced codes over trees [J3]. Since this family of codes is new, we used a lot of techniques of classical coding theory, including Hamming weight, Sphere-packing bound, Gilbert-Varshamov bound, etc. We also used several combinatorial techniques such as, graph theory, generator functions, recursion and mathematical induction, in order to calculate the sizes of balls of trees. The last construction of this paper was found using a computer search algorithm, and was proved using algebraic methods.

Lastly, in [C5] we used Hadamard codes and some techniques on graph theory, including the construction algorithm, and combinatorial techniques for constructing functional $k$-batch codes.
Part I

Published Journal Papers
Chapter 3

Codes over Graphs

Lev Yohananov and Eitan Yaakobi

Abstract

Motivated by systems where the information is represented by a graph, such as neural networks, associative memories, and distributed systems, we present in this work a new class of codes, called codes over graphs. Under this paradigm, the information is stored on the edges of undirected or directed complete graphs, and a code over graphs is a set of graphs. A node failure is the event where all edges in the neighborhood of the erased node have been erased. We say that a code over graphs can tolerate $\rho$ node failures if it can correct the erased edges of any $\rho$ failed nodes in the graph. While the construction of optimal codes over graphs can be easily accomplished by MDS codes, their field size has to be at least $O(n^2)$, when $n$ is the number of nodes in the graph. In this work we present several constructions of codes over graphs with smaller field size. To accomplish this task we use constructions of product codes and rank metric codes. Furthermore, we present optimal codes over graphs correcting two node failures over the binary field, when the number of nodes in the graph is a prime number. Lastly, we also provide upper bound on the number of nodes for optimal codes.

3.1 Introduction

The traditional setup to represent information is by a vector over some fixed alphabet. Although this commonly used model is the most practical one, especially for storage and communication applications, it does not necessarily fit all information systems. In this work we study a different approach where the information is represented by a graph. This model is motivated by several information systems. For example, in neural networks, the neural units are connected via links which store and transmit information between the neural units [4]. Similarly, in associative memories, the information is stored by associations between different
data items \[20\]. These two examples mimic the brain functionality which stores and processes information by associations between the information units. Furthermore, representing information in a graph can model a distributed storage systems \[1\] while every two nodes can share a link with the information that is stored between the nodes. For example, a node may correspond to a user and the edges are files which are shared between two users, while self loops are the user’s files.

A set of graphs called, code over graphs, is a new class of codes storing the information on the edges of the graph. In other words, each codeword of such a code is a graph with \( n \) nodes (vertices) and each edge stores a symbol over some fixed alphabet. There are two families of such codes. The first family consists of undirected complete graphs with self loops. In this case, the information is stored on each of the \( \binom{n+1}{2} \) edges. Similarly, the second family consists of directed complete graphs with self loops and the information is stored on the \( n^2 \) edges of the graph. A node failure is the event where all the edges in the node’s neighborhood have been erased, and the goal of this work is to construct codes over graphs that can efficiently correct node failures. Namely, we say that a code over graphs can correct \( \rho \) node failures if it is possible to correct the erased edges in the neighborhoods of any \( \rho \) failed nodes. We study node failures since they correspond to the events of failing neural units in a neural network, data loss in an associative memory, and unavailable or failed nodes in distributed storage systems. In case every node corresponds to a user, a node failure implies that the user’s files as well as the ones that are shared with the user are erased. Furthermore, the information stored in a complete graph can be represented by an \( n \times n \) array and a failure of the \( i \)th node corresponds to the erasure of the \( i \)th row and \( i \)th column in the array. Hence, this problem is translated to the problem of correcting symmetric crisscross erasures in square symmetric or non-symmetric arrays \[14, 16, 17\].

Assume a code over undirected graphs with \( n \) nodes such that every edge stores a symbol. If \( \rho \) nodes have failed then the number of edges that were erased is

\[
\binom{n+1}{2} - \binom{n-\rho+1}{2} = n\rho - \binom{\rho}{2}.
\]

Therefore, according to the Singleton bound, the number of redundancy edges for every code which tolerates \( \rho \) node failures is at least \( n\rho - \binom{\rho}{2} \). Similarly, for the directed case, the failure of any \( \rho \) nodes translated to \( 2n\rho - \rho^2 \) failed edges in the graph and thus the minimum number of redundancy edges of such a code is at least

\[
n^2 - (n-\rho)^2 = 2n\rho - \rho^2.
\]

A code over undirected, directed graphs which meets the lower bound \[3.1\], \[3.2\] on the number of redundancy edges, respectively, will be called an optimal code over graphs. While the construction of optimal codes meeting these bounds can be easily accomplished by MDS codes, their field size has to be at least \( \mathcal{O}(n^2) \).
Our main goal in this work is the construction of codes over graphs with smaller fields.

Since every graph can be represented by its adjacency matrix, a natural approach to construct codes over graphs is by their adjacency matrices. Thus, this class of codes is quite similar to the class of array codes, such as maximum-rank array codes [14], d-codes [15, 17], B-codes [13], EVENODD codes [2], STAR codes [8], RDP code [23], X-codes [11], and regenerating codes [15, 18, 6, 12, 21, 22]. However, there are several differences between classical array codes and codes over graphs. First, the adjacency matrix of a graph is a square matrix. Second, when the graphs are undirected, the adjacency matrices are symmetric. Third, a failure of the $i$th node in the graph corresponds to the failure of the $i$th row and the $i$th column in the adjacency matrix.

Most existing constructions of array codes are not designed for symmetric arrays, and they do not support this special row–column failure model. However, it is still possible to use existing code constructions and modify them to the special structure of the above erasure model in graphs. There are several candidates for this approach, such as product codes [1, 5], rank-metric codes [14, 16, 17], and variants of EVENODD codes [2].

The rest of this paper is organized as follows. In Section 3.2, we formally define the graph models studied in this paper and some preliminary results. In Section 3.3, we present codes over graphs correcting arbitrary number of node failures over a field of size at least $n - 1$. In Section 3.4, we present binary non-optimal constructions with respect to the bound in (3.3) and (3.4). Our main result in the paper, presented in Section 3.5, is an optimal binary code over undirected graphs correcting two node failures, when the number of nodes is prime. Then, in Section 3.6, we show how to extend this construction for directed graphs. Lastly, in Section 3.7, we study bounds on the existence of optimal codes over graphs correcting $\rho$ node failures. Section 3.8 concludes the paper.

### 3.2 Definitions and Preliminaries

In this section we formally define the tools and the definitions used throughout the paper. For a positive integer $n$, the set $\{0, 1, \ldots, n - 1\}$ will be denoted by $[n]$. For a prime power $q$, $\mathbb{F}_q$ is a finite field of size $q$. A linear code of length $n$ and dimension $k$ over $\mathbb{F}_q$ will be denoted by $[n, k]_q$ or $[n, k, d]_q$, where $d$ denotes its minimum distance.

We will denote a graph by $G = (V_n, E)$, where $V_n = \{v_0, v_1, \ldots, v_{n-1}\}$ is the set of $n$ nodes (vertices) and $E \subseteq V_n \times V_n$ is its edge set. In this paper, we only study complete graphs with self loops that can be directed or undirected. The edge set of a directed graph $G$ over $\Sigma$ will be defined by $E = V_n \times V_n$, with a labeling function $L_D : V_n \times V_n \rightarrow \Sigma$. We will use the notation $G = (V_n, L_D)$ for such graphs, since we can fully characterize the graph $G$ by its vertex set $V_n$ and its labeling function $L_D$. Similarly, the edge set of an undirected graph $G$ over $\Sigma$
will be defined by \( E = \{(v_i, v_j) \mid (v_i, v_j) \in V_n \times V_n, i \geq j\} \), with a labeling function \( L_U : V_n \times V_n \rightarrow \Sigma \) and we will use the notation \( G = (V_n, L_U) \) for such graphs. By a slight abuse of notation, every undirected edge in the graph will be denoted by \( \langle v_i, v_j \rangle \) where the order in this pair does not matter, that is, the notation \( \langle v_i, v_j \rangle \) is identical to the notation \( \langle v_j, v_i \rangle \). Note that a directed, undirected graph with \( n \) vertices has \( n^2, \binom{n+1}{2} \) edges, respectively. A graph \( G = (V_n, L) \) is a general definition that refers to both directed and undirected graphs, while it will be clear from the context which case the notation refers to.

The **adjacency matrix** of a graph \( G = (V_n, L) \) is an \( n \times n \) matrix over \( \Sigma \) denoted by \( A_G = [a_{i,j}]_{i,j=0}^{n-1} \), where \( a_{i,j} = L(v_i, v_j) \) for a directed graph, and \( a_{i,j} = L(v_i, v_j) \) for an undirected graph, while \( i,j \in [n] \). Notice that the adjacency matrix of an undirected graph is symmetric. For undirected graphs, we also define the **lower-triangle-adjacency matrix** of \( G \) to be the \( n \times n \) matrix \( A'_G = [a'_{i,j}]_{i,j=0}^{n-1} \) such that \( a'_{i,j} = a_{i,j} \) if \( i \geq j \) and otherwise \( a'_{i,j} = 0 \). The **upper-triangle-adjacency matrix** is defined similarly. We define the zero graph by \( G_0 \) if for all \( i,j \in [n] \), we have \( a_{i,j} = 0 \).

The next example demonstrates the above definitions for undirected graphs.

**Example 1.** Let \( G \) be a complete undirected graph with self loops over \( \mathbb{F}_2 \) and let \( V_4 = \{v_0, v_1, v_2, v_3\} \) be its node set. The graph \( G \), its adjacency matrix \( A_G \), and lower-triangle-adjacency matrix \( A'_G \) are shown in Fig. 3.1 where the edges \( \langle v_0, v_1 \rangle, \langle v_0, v_2 \rangle, \langle v_0, v_3 \rangle \) and \( \langle v_1, v_2 \rangle \) are labeled with 1 and the rest of the edges are labeled with 0.

![Figure 3.1: The undirected graph G, its adjacency matrix AG, and its lower-triangle-adjacency matrix A'_G.](image)

Let \( \Sigma \) be a ring and \( G_1 \) and \( G_2 \) be two graphs over \( \Sigma \) with the same node set \( V \). The operator “+” between \( G_1 \) and \( G_2 \) over \( \Sigma \), is defined by \( G_1 + G_2 = G_3 \), where \( G_3 \) is the unique graph satisfying \( A_G_1 + A_G_2 = A_G_3 \). Similarly, the operator “.” between \( G_1 \) and an element \( \alpha \in \Sigma \), is denoted by \( \alpha \cdot G_1 = G_3 \), where \( G_3 \) is the unique graph satisfying \( \alpha \cdot A_G_1 = A_G_3 \).

**Definition 1.** Let \( V_n \) be the set of nodes \( V_n = \{v_0, \ldots, v_{n-1}\} \). A code over directed graphs over \( \Sigma \) of length \( n \) and size \( M \) is a set of directed graphs \( \mathcal{C}_D = \{G_i = (V_n, L_{D_i}) \mid i \in [M]\} \) over \( \Sigma \), denoted by \( \mathcal{D}(n,M)_\Sigma \). Similarly a code
over undirected graphs} over \( \Sigma \) of length \( n \) and size \( M \) is a set of undirected graphs \( \mathcal{C}_U = \{G_i = (V_n, L_U) | i \in [M]\} \) over \( \Sigma \), denoted by \( \mathcal{U}(n, M)_\Sigma \). In case that \( \Sigma = \{0, 1\} \), the directed and the undirected codes over graphs will simply be denoted by \( \mathcal{D}(n, M) \) and \( \mathcal{U}(n, M) \). A code over graphs \( \mathcal{C}_D \) is a general definition that refers to both codes over directed graphs and codes over undirected graphs, while the meaning will be clear from the context.

The {definition} of a code over directed, undirected graphs \( \mathcal{C}_D, \mathcal{C}_U \) is \( k_D = \log_{|\Sigma|} M, k_U = \log_{|\Sigma|} M \), respectively. The {rate} of a code over directed, undirected graphs is \( R_D = k_D/n^2, R_U = k_U/(n+1)^2 \) and the {redundancy} is defined to be \( r_D = n^2 - k_D, r_U = (n+1)^2 - k_U \), respectively.

A code over directed graphs \( \mathcal{C}_D \) over a ring \( \Sigma \) will be called linear if for every \( G_1, G_2 \in \mathcal{C}_D \) and \( \alpha, \beta \in \Sigma \) it holds that \( \alpha G_1 + \beta G_2 \in \mathcal{C}_D \). A linear code over undirected graphs is defined similarly. We denote such codes over directed, undirected graphs by \( \mathcal{D}([n], k_D)_\Sigma, \mathcal{U}([n], k_U)_\Sigma \), respectively.

A linear code over directed, undirected graphs will be called {systematic} if the first \( k \) nodes contain the \( k^2, (k^2+1) \) unmodified information symbols on their edges, respectively. All other \( n^2 - k^2, (n^2+1) - (k^2+1) \) edges in the graph are called {redundancy edges}, respectively. In this case we say that there are \( k \) {information nodes} and \( r = n-k \), {redundancy nodes}. The number of {information edges} is \( k_D = k^2, k_U = (k^2+1) \), the redundancy is \( r_D = n^2 - k^2, r_U = (n^2+1) - (k^2+1) \), and the rate is \( R_D = k^2/n^2, R_U = (k^2+1)/(n^2+1) \) for directed, undirected codes over graphs, respectively. We denote such a code by \( \mathcal{SD}([n], k)\Sigma \) for directed codes over graphs and \( \mathcal{SU}([n], k)\Sigma \) for undirected codes over graphs.

**Definition 2.** Let \( G = (V_n, L_D) \) be a directed graph. For \( i \in [n] \), the {out-neighborhood edge set}, {in-neighborhood edge set}, of the \( i \)-th node is defined to be the set

\[
N_i^{\text{out}} = \{(v_i, v_j) \mid j \in [n]\}, \quad N_i^{\text{in}} = \{(v_j, v_i) \mid j \in [n]\},
\]

respectively, and the {neighborhood edge set} of the \( i \)-th node is the set \( N_i = N_i^{\text{out}} \cup N_i^{\text{in}} \). Note that the \( i \)-th out-neighborhood, in-neighborhood edge set, corresponds to the \( i \)-th row, column, in an adjacency matrix \( A_G \), respectively, and the \( i \)-th neighborhood edge set is the union of the \( i \)-th column and the \( i \)-th row in the adjacency matrix. Similarly, the neighborhood edge set of the \( i \)-th node of an undirected graph \( G = (V_n, L_U) \) is defined by \( N_i = \{(v_i, v_j) \mid j \in [n]\} \), which corresponds to the \( i \)-th column and row in an adjacency matrix \( A_G \).

The {node failure} of the \( i \)-th node is the event in which all the edges in the neighborhood set of the \( i \)-th node, i.e. \( N_i \), are erased. We will also denote this edge set by \( F_i \) and refer to it by the {failure set} of the \( i \)-th node. For convenience, for directed graphs we also define the {out-failure set}, {in-failure set} of the \( i \)-th node by \( F_i^{\text{out}} = N_i^{\text{out}}, F_i^{\text{in}} = N_i^{\text{in}} \), respectively.

A code over directed graphs is called a directed \( \rho \)-node-erasure-correcting code if it can correct any failure of at most \( \rho \) nodes in each of its graphs. An
**undirected ρ-node-erasure-correcting code** is defined similarly. A **ρ-node-erasure-correcting code** is a general definition that refers to both codes over directed and undirected graphs.

According to the Singleton bound, we deduce that the minimum redundancy \( r_D, r_U \) of any directed, undirected \( ρ \)-node-erasure-correcting code of length \( n \), satisfies

\[
\begin{align*}
r_D &\geq n^2 - (n - ρ)^2 = 2nρ - ρ^2, \\
r_U &\geq \left(\frac{n + 1}{2}\right) - \left(\frac{n - ρ + 1}{2}\right) = nρ - \left(\frac{ρ}{2}\right),
\end{align*}
\]

(3.3)  
(3.4)

respectively. A code over directed, undirected graphs satisfying the first, second inequality with equality will be called **optimal**, respectively. In this paper we study only linear codes. We also state that every optimal code according to the bound (3.3) or (3.4) is systematic as it was defined above. Hence, for systematic codes over graphs the number of redundancy nodes is at least \( ρ \). Note that for all \( n \) and \( ρ \), one can always construct an optimal directed \( ρ \)-node-erasure-correcting code from an \([n^2, (n - ρ)^2, 2nρ - ρ^2 + 1]\) MDS code. Similarly, one can always construct an optimal undirected \( ρ \)-node-erasure-correcting code from an \([\binom{n+1}{2}, \binom{n-ρ+1}{2}, nρ - \left(\frac{ρ}{2}\right) + 1]\) MDS code. However, in both cases the field size of the code over graphs will be at least \( Θ(n^2) \). Our goal in this work is to construct \( ρ \)-node-erasure-correcting codes over smaller fields. When possible, we seek the field size to be binary and in any event at most \( O(n) \).

The next example exemplifies the definitions of codes over undirected graphs.

**Example 2.** The following codes given in Fig. 3.2 are systematic binary single-node-erasure-correcting codes of length 3. The left figure illustrates an undirected code over graphs, and the right figure illustrates a directed code over graphs. Both constructions store the information on edges of the complete subgraph of nodes \( v_0 \) and \( v_1 \). For the undirected case, the neighborhood set of each node belongs to a simple parity code of length 3. Similarly for the directed case, the out-neighborhood set and the in-neighborhood set of each node belongs to a simple parity code of length 3.

The code construction from Example 2 is easily extended for arbitrary number of nodes. For undirected graphs this construction is formulated as follows.

**Construction 1.** Let \( n \geq 2 \) be a positive integer. The code over undirected graphs \( C_U \) is defined as follows.

\[
C_U = \{G = (V_n, L_U) : \forall i \in [n], \sum_{j=0}^{n-1} L_U(v_i, v_j) = 0\}.
\]

As mentioned above, the constraints imposed in this constructions state that the edges in the neighborhood of each vertex in the graph form a simple parity
code, so we call it a *neighborhood constraint*. In the adjacency matrix, that means that each row and column belongs to a simple parity code. The correctness of this construction is proved in the following theorem.

**Theorem 3.** The code $C_{U_i}$ is an optimal binary undirected single-node-erasure-correcting code $SU-[n, n-1]$.

**Proof.** Suppose that the node $v_i$ is erased. Then, for each node $v_s$ such that $v_s \neq v_i$, the edge $\langle v_i, v_s \rangle$ is corrected by

$$L_{U}(v_i, v_s) = \sum_{j=0, j \neq i}^{n-1} L_{U}(v_s, v_j),$$

and the only uncorrected edge left is the self loop $\langle v_i, v_i \rangle$. Therefore, the edge $\langle v_i, v_i \rangle$ is corrected by

$$L_{U}(v_i, v_i) = \sum_{j=0, j \neq i}^{n-1} L_{U}(v_i, v_j).$$

Notice that the code $C_{U_i}$ is of size $\left(\begin{array}{c} n \\ 2 \end{array}\right)$, so its redundancy satisfies the bound in (3.4) with equality and thus the code is optimal.

The construction of an optimal binary directed single-node-erasure-correcting code $SD-[n, n-1]$ is similar.

Next we define a distance metric over graphs that will be used in the construction of codes correcting node failures.
Definition 4. Let $G = (V_n, L)$ be a graph and let $E$ be the set of all nonzero labeled edges of $G$, i.e., $E = \{e \in V_n \times V_n \mid L(e) \neq 0\}$. A vertex cover $W$ of $G$ is a subset of $V_n$ such that for each $(v_i, v_j) \in E$ (or $(v_i, v_j) \in E$ in an undirected case) either $v_i \in W$ or $v_j \in W$. Then, the graph weight of $G$ is defined by

$$w_G(G) = \min_{W \text{ is a vertex cover of } G} \{|W|\}.$$  

Intuitively, the value $w_G(G_1-G_2)$ is simply the minimum number of nodes whose removals make $G_1$ and $G_2$ identical. The graph distance between two graphs $G_1, G_2$ will be denoted by $d_G(G_1, G_2)$ and it holds that $d_G(G_1, G_2) = w_G(G_1-G_2)$.

Lemma 5. The graph distance is a metric.

Proof. 1. Clearly, $d_G(G_1, G_2) \geq 0$ since the vertex cover is defined to be non-negative, and by definition of the graph weight $d_G(G_1, G_2) = 0$ if and only if $G_1 = G_2$.

2. Symmetry: $d_G(G_1, G_2) = w_G(G_1 - G_2) = w_G(G_2 - G_1) = d_G(G_2, G_1)$, since each edge of the graph $G = (G_1 - G_2)$ has a non-zero label if and only if it has a non-zero label in the graph $G' = (G_2 - G_1)$.

3. The triangle inequality:

$$d_G(G_1, G_2) = w_G(G_1 - G_2) = w_G(G_1 - G_3 + G_3 - G_2) = \min_{W \text{ v.c. of } (G_1 - G_3) + (G_3 - G_2)} \{|W|\} \leq \min_{W \text{ v.c. of } G_1 - G_3} \{|W|\} + \min_{W \text{ v.c. of } G_3 - G_2} \{|W|\} = w_G(G_1 - G_3) + w_G(G_3 - G_2) = d_G(G_1, G_3) + d_G(G_3, G_2),$$

where the inequality holds since each non-zero labeled edge of the graph $G = ((G_1 - G_3) + (G_3 - G_2))$ is a non-zero labeled edge of either the graph $G' = (G_1 - G_3)$ or $G'' = (G_3 - G_2)$.

The minimum distance of a code over graphs $C_G$, denoted by $d(C_G)$, is a minimum graph distance between any two distinct graphs in $C_G$, that is

$$d(C_G) = \min_{G_1 \neq G_2} \min_{G_1, G_2 \in C_G} \{d_G(G_1, G_2)\},$$

and in case the code is linear

$$d(C_G) = \min_{G \in C_G, G \neq G_0} \{w_G(G)\}.$$  

Theorem 6. A linear code over graphs $C_G$ is a $\rho$-node-erasure-correcting code if and only if its minimum distance satisfies $d(C_G) \geq \rho + 1$.  

27
Proof. Assume that the minimum distance of $C_G$ satisfies $d(C_G) \geq \rho + 1$. Let $Z$ be a received graph after having $\rho$ node failures of a graph $G \in C_G$. Let $D$ be a decoder of $C_G$, $D(Z) \in C_G$ such that $D(Z)$ is consistent with $Z$ on all non-erased edges. We will show that $D(Z) = G$ is the unique solution tolerating $\rho$ node failures. Assume that there are two different graphs $G_1, G_2 \in C_G$ that are consistent with $Z$ on all the non-erased edges. Therefore, we have $w_G(G_1 - G_2) = d_G(G_1, G_2) \leq \rho \leq d(C_G) - 1$, since all of the edges of the graph $G' = (G_1 - G_2)$ can be covered only by the failed nodes. Thus, we get a contradiction.

Next, suppose that $C_G$ has a minimum distance less than $\rho + 1$. We will show that there is no decoder that can correct any $\rho$ node failures. Assume that there are two distinct graphs $G_1, G_2 \in C_G$ such that $d_G(G_1, G_2) \leq d(C_G) - 1 < \rho$. Denote by $G$ the graph $G_1 - G_2$, where $w_G(G) = w_G(G_1 - G_2) = d_G(G_1, G_2)$, so $w_G(G) < \rho$. Let $W$ be a vertex cover of $G$ and assume that the nodes of $W$ in $G_1$ were erased. Since $G = (G_1 - G_2)$, the graphs $G_1$ and $G_2$ are consistent on all the non-erased edges of $G_1$, and the decoder of $C_G$ will not be able to correct such node failures. □

For the convenience of the reader, relevant notation and terminology referred to throughout the paper is summarized in Table 3.1.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>The number of nodes</td>
<td>Sec. 3.2</td>
</tr>
<tr>
<td>$G = (V, E)$</td>
<td>An undirected complete graph over $n$ nodes</td>
<td>Sec. 3.2</td>
</tr>
<tr>
<td>$A_G$</td>
<td>The adjacency matrix of a graph $G$</td>
<td>Sec. 3.2</td>
</tr>
<tr>
<td>$L(v_i, v_j)$</td>
<td>The label of the edge $\langle v_i, v_j \rangle$</td>
<td>Sec. 3.2</td>
</tr>
<tr>
<td>$D_{\Sigma}(n, M)$, $C_D$</td>
<td>A code over directed graphs of length $n$ and size $M$</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$U_{\Sigma}(n, M)$, $C_U$</td>
<td>A code over undirected graphs of length $n$ and size $M$</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$D_{\Sigma}[n, M]$</td>
<td>A linear code over directed graphs of length $n$ and size $M$</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$U_{\Sigma}[n, M]$</td>
<td>A linear code over undirected graphs of length $n$ and size $M$</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$k_D$</td>
<td>The dimension of a code over directed graphs</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$k_U$</td>
<td>The dimension of a code over undirected graphs</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$R_D$</td>
<td>The rate of a code over directed graphs</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$R_U$</td>
<td>The rate of a code over undirected graphs</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$r_D$</td>
<td>The redundancy of a code over directed graphs</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$r_U$</td>
<td>The redundancy of a code over undirected graphs</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$SD_{\Sigma}[n, k]$</td>
<td>Systematic code over directed graphs</td>
<td>Def. 2</td>
</tr>
<tr>
<td>$SU_{\Sigma}[n, k]$</td>
<td>Systematic code over undirected graphs</td>
<td>Def. 2</td>
</tr>
<tr>
<td>$d(C_G)$</td>
<td>The minimum distance of a code over graphs $C_G$</td>
<td>Def. 1</td>
</tr>
<tr>
<td>$S_h$</td>
<td>The neighborhood parity constraint</td>
<td>Eq. 3.5</td>
</tr>
<tr>
<td>$D_m$</td>
<td>The diagonal parity constraint</td>
<td>Eq. 3.6</td>
</tr>
</tbody>
</table>
3.3 Optimal Multiple-Node-Erasure-Correcting Codes over Linear Field Size

In the previous section we saw that optimal $\rho$-node-erasure-correcting codes are easy to construct over a field of size $\Theta(n^2)$. In this section we will present two constructions that reduce the large field to $\Theta(n)$. Namely, we show an optimal construction of directed, undirected $\rho$-node-erasure-correcting codes $S^D-[n,n-\rho]_{F_q}$, $SU-[n,n-\rho]_{F_q}$ for all $n$ and $\rho$, where $q$ is a prime power greater than $n-1$, respectively. This result is developed using the construction of product codes that were introduced by Elias in [5] and was discussed by Abramson in [1].

3.3.1 Optimal Codes over Directed Graphs

We first review the construction of product codes. Let $C_1, C_2$ be a linear code with parameters $[n_1, k_1, d_1]_q, [n_2, k_2, d_2]_q$, respectively. Denote by $H_1, H_2$ the parity-check matrix of $C_1, C_2$, respectively. Then, the product code of $C_1, C_2$, denoted by $P(C_1, C_2)$, is defined by

$$P(C_1, C_2) = \{ A \in \mathbb{F}_q^{n_1 \times n_2} \mid H_1 A = A H_2^\top = 0 \}.$$  

It was shown in [11] that $P(C_1, C_2)$ is an $[N, K, D]_q$ linear code with $N = n_1 n_2, K = k_1 k_2$ and $D = d_1 d_2$. Therefore, according to the definition of $P(C_1, C_2)$, each column of $A \in P(C_1, C_2)$ is a codeword of $C_1$ and each row of $A \in P(C_1, C_2)$ is a codeword of $C_2$. In case where $C_1 = C_2 = C$, the code is denoted by $P(C)$. If $C$ is an $[n, k, d]_q$ code, then $P(C) = \{ A \in \mathbb{F}_q^{n \times n} \mid H A = A H^\top = 0 \}$ is an $[N, K, D]$ linear code with $N = n^2, K = k^2$ and $D = d^2$, where each row and each column of $A \in P(C)$ is a codeword of $C$.

We are now ready to present a construction of optimal $\rho$-node-erasure-correcting codes over directed graphs with a field size not smaller than $n-1$. Let us consider the adjacency matrix of each graph in the code in order to explain the main idea of the construction. Each row and column in the adjacency matrix belongs to an $[n,n-\rho,\rho+1]_q$ MDS code, which will be denoted by $C$, where $q \geq n-1$. Equivalently, each in-neighborhood edge set and out-neighborhood edge set of every node $v, v \in V_n$ is a codeword in $C$. This construction is formalized as follows.

Construction 2. Let $n$ and $\rho$ be two positive integers such that $\rho < n$. Let $C$ be an $[n,n-\rho,\rho+1]_q$ MDS code, for $q \geq n-1$, and let $P(C)$ be its product code. The code $C_{D_1}$ is defined as follows,

$$C_{D_1} = \{ G = (V_n, L_D) \mid A_G \in P(C) \}.$$  

The correctness of Construction 2 is proved in the next theorem.

Theorem 7. For all $\rho$ and $n$ such that $\rho < n$, the code $C_{D_1}$ is an optimal directed $\rho$-node-erasure-correcting code $D-[n, k_D = (n-\rho)^2]_{F_q}$, where $q \geq n-1$. 

29
Proof. As mentioned before, the dimension of $\mathcal{P}(\mathcal{C})$ is $(n - \rho)^2$, therefore $\mathcal{C}_{D_1}$ is a linear code over graphs with dimension $k_\mathcal{D} = (n - \rho)^2$. Let $G = (V_n, L_\mathcal{D})$, $G \in \mathcal{C}_{D_1}$ be a graph and assume that $\rho$ of its nodes are erased, where their indices are denoted by the set $J \subseteq [n]$. Let $U = \{v_i \in V_n \mid i \in J\}$ and $W = V_n \setminus U$. The decoding of the erased edges is invoked in two steps. In the first step, all incoming and outgoing edges of each $w \in W$ are corrected, and in the second step the remaining incoming and outgoing edges of each $u \in U$ are corrected.

1. Since every row and every column of $A_G$ is a codeword of $\mathcal{C}$, the rows and columns with at most $\rho$ erasures can be corrected. For every $i \in [n] \setminus J$ the $i$-th row and column has exactly $\rho$ erasures, and therefore the incoming and outgoing edges of each $w \in W$ are corrected.

2. For each $w \in W$ exactly one incoming edge and exactly one outgoing edge of each $u \in U$ was corrected in the first step. Since $|W| = n - \rho$, the number of uncorrected incoming and outgoing edges for each $u \in U$ is $\rho$. Therefore the incoming and outgoing neighborhoods of each $u \in U$ can be corrected by the decoder of $\mathcal{C}$ as well.

Note that the code $\mathcal{C}_{D_1}$ is also systematic where its first $n - \rho$ nodes are the information nodes. In the adjacency matrix, this corresponds to having the information symbols in the upper left $(n - \rho) \times (n - \rho)$ matrix. Then, each of the first $n - \rho$ columns encoded with the systematic MDS code $\mathcal{C}$, and then the same procedure is invoked on each of the $n$ rows. In the next section, we will construct similar codes over undirected graphs.

### 3.3.2 Optimal Codes over Undirected Graphs

The construction of optimal codes over undirected graphs can be established by taking a sub-code of the product code $\mathcal{P}(\mathcal{C})$ which consists of only symmetric matrices. Let $\mathcal{C}$ be a linear code with parameters $[n, k, d]_q$. Denote by $H$ the parity-check matrix of $\mathcal{C}$ and denote by $G$ a generator matrix of $\mathcal{C}$. Then, the **symmetric product code** of $\mathcal{C}$, denoted by $\mathcal{H}(\mathcal{C})$, is defined by

$$\mathcal{H}(\mathcal{C}) = \{A \in \mathbb{F}_q^{n \times n} \mid HA = 0, A = A^\top\}.$$ 

First we will show that $\mathcal{H}(\mathcal{C})$ is an $[N, K, D]$ linear code with $N = n^2, K = \binom{k+1}{2}$ and $D = d^2$.

**Lemma 8.** The symmetric product code $\mathcal{H}(\mathcal{C})$ is an $[N, K, D]$ linear code with $N = n^2, K = \binom{k+1}{2}$ and $D = d^2$.

**Proof.** The code $\mathcal{H}(\mathcal{C})$ is linear since it is defined by parity-check equations. Since $\mathcal{H}(\mathcal{C})$ is a product code, its minimum distance is $D = d^2$. Let $u_0, u_1, \ldots, u_{\binom{k+1}{2}-1}$ be information symbols over $\mathbb{F}_q$ that will be stored in a symmetric matrix $U^2$ that
will be called an information matrix. Each information matrix \( U \in \mathbb{F}_q^{k \times k} \), will be encoded by \( A = G^T U G \), and it is straightforward to verify that \( HA = AH^T = 0 \).

We show that if \( U = U^\top \) then,

\[
A = G^T U G = G^T U^\top G = (UG)^\top (G^T)^\top = (G^T U G)^\top = A^\top.
\]

Moreover, if \( A = A^\top \) then

\[
A = A^\top \\
G^T U G = (G^T U G)^\top \\
G^T U G = (UG)^\top (G^\top)^\top \\
G^T U G = G^\top U^\top G,
\]

and since \( G \) is a full row rank matrix, it implies that \( U = U^\top \). Therefore, the dimension of \( H(C) \) is equal to the dimension of \( \{ U \in \mathbb{F}_q^{k \times k} \mid U = U^\top \} \), that is \( \binom{k+1}{2} \).

This construction for undirected graphs is formalized as follows.

**Construction 3.** Let \( n \) and \( \rho \) be two positive integers such that \( n > \rho \). Let \( C \) be an \( [n, n - \rho, \rho + 1]_q \) MDS code, for \( q \geq n - 1 \), and let \( H(C) \) be its symmetric product code. The code \( C_{U^2} \) is defined as follows,

\[
C_{U^2} = \{ G = (V_n, L_U) \mid A_G \in H(C) \}.
\]

Notice that by Lemma 8 the dimension of code \( H(C) \) is \( \binom{n-\rho+1}{2} \), and therefore by the definition of the code \( C_{U^2} \), its dimension is \( k_{U^2} = \binom{n-\rho+1}{2} \). All other details for the correctness proof of Construction 3 are identical to the one of Theorem 7, and thus we only state here the next theorem.

**Theorem 9.** For all \( \rho \) and \( n \) such that \( \rho < n \), the code \( C_{U^2} \) is an optimal undirected \( \rho \)-node-erasure-correcting code \( U\text{-}[n, k_{U^2} = \binom{n-\rho+1}{2}]_{\mathbb{F}_q} \), where \( q \geq n - 1 \).

Similarly to the code \( C_{D_1} \), the code \( C_{U^2} \) is also systematic.

### 3.4 Binary-Node-Erasure-Correcting Codes

In this section we study binary constructions of codes over directed and undirected graphs based upon the results by Roth from [14] and Schmidt from [16, 17].

An \( [n \times n, k, \mu] \) linear array code \( C \) over a field \( \mathbb{F} \) is a \( k \)-dimensional linear subspace of \( n \times n \) matrices over \( \mathbb{F} \), where the minimum rank of all nonzero matrices in \( C \) is at least \( \mu \). It was shown in [14] that such codes can correct \( \mu - 1 \) row or column erasures. Furthermore, the bound on such array codes states that \( k \leq n(n-\mu+1) \) [14]. In this section we present non-optimal binary constructions for undirected and directed codes over graphs based upon the results from [14, 16, 17].
3.4.1 Binary Construction of Codes over Directed Graphs

A construction of binary \([n \times n, k, \mu]\) linear array codes where \(k = n(n - \mu + 1)\) and \(\mu = 2\rho + 1\) was shown in [14]. Based on these codes, we present the following construction of binary directed codes over graphs.

**Construction 4.** Let \(C\) be an \([n \times n, n(n - 2\rho), 2\rho + 1]\) binary optimal array code from [14], where \(\rho < n/2\). The code over graphs \(C_{D_3}\) is defined as follows,

\[
C_{D_3} = \{G = (V_n, L_D) \mid A_G \in C\}.
\]

Next the correctness of Construction 4 is proved.

**Theorem 10.** For all \(\rho < n/2\), the code \(C_{D_3}\) is a linear binary directed \(\rho\)-node-erasure-correcting code \(D\)-\([n, k_D = n(n - 2\rho)]\).

**Proof.** Notice that since the code \(C\) is linear, the code \(C_{D_3}\) is also linear. Let \(G\) be a graph in the code \(C_{D_3}\) and let \(A_G\) be its adjacency matrix. Assume some \(\rho\) nodes failed in \(G\). The failure of these \(\rho\) nodes corresponds to erasure of the same \(\rho\) rows and columns in \(A_G\). Since \(A_G \in C\) the minimum rank of \(A_G\) is \(2\rho + 1\), and any \(2\rho\) row or column erasures can be corrected. In particular, the erased \(\rho\) rows and \(\rho\) columns can be corrected as well, thereby correcting the \(\rho\) failed nodes.

Note that this construction does not provide optimal \(\rho\)-node-erasure-correcting codes since \(r_U = 2n\rho\), which does not meet the bound in (3.3). For example, for \(\rho = 2\) the difference between the code redundancy and the bound is 4 redundancy bits.

The construction of binary optimal array codes \([n \times n, n(n - r), r + 1]\) from [14] has also a systematic construction, where the first \(n - 2\rho\) rows of each matrix store the information bits and the last \(2\rho\) rows store the redundancy bits. Therefore, we can use this family of codes also for the construction of systematic binary \(\rho\)-node-erasure-correcting codes over directed graphs \(SD\)-\([n, k = n - 2\rho]\) for \(\rho < n/2\). In this case, the number of redundancy edges will be \(n^2 - (n - 2\rho)^2 = 4n\rho - 4\rho^2\). Therefore, the redundancy of this code is \((4n\rho - 4\rho^2) - (2n\rho - \rho^2) = 2n\rho - 3\rho^2\) far from optimality.

3.4.2 Binary Construction of Codes over Undirected Graphs

A construction of binary \([n \times n, k, \mu]\) symmetric linear array codes where

\[
k = \begin{cases} 
n(n - \mu + 2)/2, & n - \mu \text{ is even}, \\
(n + 1)(n - \mu + 1)/2, & n - \mu \text{ is odd}, 
\end{cases}
\]

was shown in [16]. Based on these codes, we present the following construction of binary undirected \(\rho\)-node-erasure-correcting codes.
Construction 5. Let $C$ be an $[n \times n, k, \mu = 2\rho + 1]$ symmetric binary array code from [10], where

$$k = \begin{cases} 
\frac{n(n - 2\rho + 1)}{2}, & n \text{ is odd}, \\
\frac{(n + 1)(n - 2\rho)}{2}, & n \text{ is even}, 
\end{cases}$$

and $\rho < n/2$. The code over graphs $C_{U4}$ is defined as follows,

$$C_{U4} = \{ G = (V_n, L_U) \mid A_G \in C \}.$$

The proof of the correctness of Construction 5 is identical to the one of Theorem 10 and is stated in the next theorem.

Theorem 11. For all $\rho < n/2$ and

$$k_{U} = \begin{cases} 
\frac{n(n - 2\rho + 1)}{2}, & n \text{ is odd}, \\
\frac{(n + 1)(n - 2\rho)}{2}, & n \text{ is even}, 
\end{cases}$$

the code $C_{U4}$ is a linear binary undirected $\rho$-node-erasure-correcting code $U\cdot [n, k_{U}]$.

This construction also does not provide optimal $\rho$-node-erasure-correcting codes since

$$r_{U} = \begin{cases} 
n\rho, & n \text{ is odd}, \\
(n + 1)\rho, & n \text{ is even}, 
\end{cases}$$

which does not achieve the bound in (3.4). For example, for $\rho = 2$ the difference between the code redundancy and the bound is one redundancy bit when $n$ is odd and three bits when $n$ is even.

In Section 3.3 we saw constructions for optimal codes over graphs over a field of size at least $n - 1$, and in this section we saw binary constructions that do not provide optimal codes over graphs. Our next task is to achieve these two properties simultaneously, that is, optimal binary codes over graphs. In the next section we show how to accomplish this task for two node failures, when the number of nodes is a prime number. The general case for arbitrary number of node failures is left for future work.

3.5 Optimal Binary Undirected Double-Node-Erasure-Correcting Codes

In this section we present a construction of binary double-node-erasure-correcting codes for undirected graphs. We use the notation $\langle a \rangle_n$ to denote the value of $(a \mod n)$.

Throughout this section we assume that $n \geq 5$ is a prime number. Let $G = (V_n, L_U)$ be a graph with $n$ vertices. Let us define for $h \in [n - 1]$

$$S_h = \begin{cases} 
\{ \langle v_h, v_\ell \rangle \mid \ell \in [n - 1] \}, & h \in [n - 2], \\
\{ \langle v_\ell, v_{n-1} \rangle \mid \ell \in [n - 1] \}, & h = n - 2.
\end{cases}$$

(3.5)
and for \( m \in [n] \)

\[
D_m = \{ (v_k, v_\ell) | k, \ell \in [n]\setminus\{n-2\}, (k+\ell)_n = m \} \cup \{ (v_{n-1}, v_{n-2}) \}.
\]  

(3.6)

Each set \( S_h \) where \( h \in [n-2] \), will be used to represent the parity constraint on the neighborhood of node \( v_h \), which correspond to row and column \( h \) in the adjacency matrix \( A_G \). Similarly, for \( m \in [n] \), each set \( D_m \) will represent parity constraints on the diagonals of \( A_G \). We first show the following properties on the sets \( S_h \) and \( D_m \), which their proof appear in \textbf{Appendix A}.

**Claim 1.** For all \( h \in [n-1], |S_h| = n - 1 \) and for all \( m \in [n], |D_m| = \frac{n+1}{2} \).

**Example 3.** The sets \( S_h, D_m \) for \( n = 7 \) are marked in Fig. 3. Note that the entries on lines with the same color belong to the same parity constraints.

![Diagram](image.png)

(a) Neighborhood Parity Paths  
(b) Diagonal Parity Paths

Figure 3.3: The neighborhood and the diagonal sets.

Recall that for \( t \in [n] \) the failure set \( F_t \) of the \( t \)-th node is its neighborhood set which we denote by \( F_t = \{ (v_\ell, v_t) | \ell \in [n] \} \). The following connections between the sets \( S_h, D_m, F_t \) hold and will be used in the correctness of the construction we present in this section. The following claim will be in use in the proof of Theorem 12 and its proof is given in \textbf{Appendix B}.

**Claim 2.** The sets \( S_h, D_m, F_t \) satisfy the following properties.

(a) For all distinct \( h, i \in [n-2], S_h \cap F_i = \{ (v_h, v_i) \} \).

(b) For all \( i \in [n-2], S_{n-2} \cap F_i = \{ (v_i, v_i) \} \).

(c) For all pairwise distinct \( i, j, h \in [n-2], S_h \cap (F_i \cup F_j) = \{ (v_h, v_i), (v_h, v_j) \} \).

(d) For all distinct \( i, j \in [n-2], S_{n-2} \cap (F_i \cup F_j) = \{ (v_i, v_i), (v_j, v_j) \} \).

(e) For all \( i \in [n-2], D_{(i-2)_n} \cap F_i = \emptyset \) and for \( i = n - 1, D_{(n-3)_n} \cap F_{n-1} = \{ (v_{n-2}, v_{n-1}) \} \).
(f) For all $i \in [n-2]$, $s \in [n] \setminus \{i-2\}$, $D_s \cap F_i = \{(v_{(s-i)n}, v_i)\}$. For $i$ such that $i = n-1$, $D_s \cap F_{n-1} = \{(v_{(s+1)n}, v_{n-1}), (v_{n-2}, v_{n-1})\}$

(g) For all distinct $i, j \in [n-2]$, $D_{(i+j)n} \cap F_j = \{v_i, v_j\}$.

(h) For all distinct $i, j \in [n-2]$, $D_{(j-2)n} \cap (F_i \cup F_j) = \{(v_{(j-i-2)n}, v_i)\}$

(i) For all distinct $i, j \in [n-2]$, $D_{(i+j)n} \cap (F_i \cup F_j) = \{v_i, v_j\}$.

(j) For all $i \in [n-2]$, $D_{(i-2)n} \cap (F_i \cup F_{n-2}) = \{(v_{i-1}, v_{n-2})\}$.

We are now ready to present the construction of optimal binary undirected double-node-erasure-correcting codes.

Construction 6. For all $n \geq 5$ prime number, let $C_{4b}$ be the following code.

$$C_{4b} = \left\{ G = (V_n, L_U) \mid \begin{cases} \text{(a)} & \sum_{(v_i, v_j) \in S_h} L_U(v_i, v_j) = 0, h \in [n-1] \\ \text{(b)} & \sum_{(v_i, v_j) \in D_m} L_U(v_i, v_j) = 0, m \in [n] \end{cases} \right\}$$

Note that in this binary construction we have two sets of constraints, (a) and (b). The first set has $n - 1$ constraints and we call each one of them constraint $S_h$, $h \in [n-1]$. Similarly, the second set has $n$ constraints that will be called constraint $D_m$, $m \in [n]$. Note that the edge $\langle v_{n-1}, v_{n-2} \rangle$ appears in each of the diagonal sets, and therefore for $m \neq n - 3$ the constraints $D_m$ are dependent on the constraint $D_{n-3}$. We will need that in order to have successful decoding when the failed nodes are $i \in [n-2]$ and $j = n - 2$, as will be shown for this case in the proof. Lastly, the correctness of this construction could be proved using Theorem 3 by showing that the minimum graph distance of this code is three, however, this will not provide a decoding algorithm as we present in the following proof.

Theorem 12. The code $C_{4b}$ is an optimal binary undirected double-node-erasure-correcting code.

Proof. Assume that nodes $i, j \in [n]$, where $i < j$ are the failed nodes. We distinguish between the following three cases.

Case 1: $i \in [n-1], j = n-1$. Using the $S_h$ constraints, $h \in [n-1] \setminus \{i\}$, the edge set $F_i \setminus \{(v_i, v_{n-2}), (v_i, v_{n-1})\}$ can be corrected by

$$L_U(v_i, v_h) = \sum_{\langle v_k, v_h \rangle \in S_h \setminus \{(v_i, v_h)\}} L_U(v_k, v_h) : h \neq n-2,$$

$$L_U(v_i, v_i) = \sum_{\langle v_k, v_i \rangle \in S_{n-2} \setminus \{(v_i, v_i)\}} L_U(v_k, v_i) : h = n-2,$$

since for $h \neq i, n - 2$ by Claim 2, $S_h \cap F_i = \{v_h, v_i\}$, and for $h = n - 2$ by Claim 2, $S_{n-2} \cap F_i = \{v_i, v_i\}$. The constraint $S_i$ then can be used to correct $\langle v_i, v_{n-2} \rangle$ by

$$L_U(v_i, v_{n-2}) = \sum_{\langle v_k, v_i \rangle \in S_i \setminus \{(v_i, v_{n-2})\}} L_U(v_k, v_i).$$
Notice that \( F_{n-1} \) is the set of all the uncorrected edges left. By Claim \( \text{2(f)} \), \( D_{n-3} \cap F_{n-1} = \{ (v_{n-2}, v_{n-1}) \} \), so we first use the constraint \( D_{n-3} \) to correct the edge \( (v_{n-2}, v_{n-1}) \) by
\[
L_U(v_{n-2}, v_{n-1}) = \sum_{(v_k, v_l) \in D_{n-3} \setminus \{ (v_{n-2}, v_{n-1}) \}} L_U(v_k, v_l).
\]
By Claim \( \text{2(f)} \) for \( m \in [n] \setminus \{ n - 3 \} \),
\[
D_m \cap F_{n-1} = \{ (v_{(m+1)n}, v_{n-1}), (v_{n-2}, v_{n-1}) \},
\]
and since the edge \( (v_{n-2}, v_{n-1}) \) is corrected, the \( D_m \) constraints can be used. Therefore, the remaining edges of the set \( F_{n-1} \) are corrected by
\[
L_U(v_{(m+1)n}, v_{n-1}) = \sum_{(v_k, v_l) \in D_m \setminus \{ (v_{(m+1)n}, v_{n-1}) \}} L_U(v_k, v_l),
\]
and that finishes the decoding of this case.

**Case 2:** \( i \in [n - 2], j = n - 2 \). By Claim \( \text{2(f)} \) \( D_{(i-2)n} \cap (F_i \cup F_{n-2}) = \{ (v_{n-1}, v_{n-2}) \} \), so using the constraint \( D_{(i-2)n} \), the edge \( (v_{n-1}, v_{n-2}) \) is corrected by
\[
L_U(v_{n-1}, v_{n-2}) = \sum_{(v_k, v_l) \in D_{(i-2)n} \setminus \{ (v_{n-1}, v_{n-2}) \}} L_U(v_k, v_l).
\]
By Claim \( \text{2(f)} \) for all \( m \in [n] \setminus \{ (i-2)n \} \), \( D_m \cap F_i = \{ (v_{(m-i)n}, v_i) \} \), and since \( (v_{n-1}, v_{n-2}) \) is the only edge that intersects between constraints \( D_m \) and \( F_{n-2} \), \( D_m \cap (F_i \cup F_{n-2}) = \{ (v_{(m-i)n}, v_i), (v_{n-1}, v_{n-2}) \} \). Since edge \( (v_{n-1}, v_{n-2}) \) is corrected, the edges in the set \( F_i \setminus \{ (v_i, v_{n-2}) \} \) are corrected by the constraints \( D_m \) as follows,
\[
L_U(v_{(m-i)n}, v_i) = \sum_{(v_k, v_l) \in D_m \setminus \{ (v_{(m-i)n}, v_i) \}} L_U(v_k, v_l).
\]
Notice that \( F_{n-2} \setminus \{ (v_{n-2}, v_{n-1}) \} \) is the set of all the uncorrected edges left. Thus, it is corrected using the \( S_h \) constraints, \( h \in [n - 1], \) similarly to the first case.

**Case 3:** \( j < n - 2 \). In this case we show an explicit algorithm which decodes all the erased edges. First, we denote the **single parity syndromes** for \( h \in [n - 1] \setminus \{ i, j \} \) by
\[
\hat{S}_h = \sum_{(v_k, v_l) \in S_h \setminus (F_i \cup F_j)} L_U(v_k, v_l),
\]
and the **diagonal parity syndromes** for \( m \in [n] \) by
\[
\hat{D}_m = \sum_{(v_k, v_l) \in D_m \setminus (F_i \cup F_j)} L_U(v_k, v_l).
\]
Let \( d = (j-i)_n, x = (-1 - d^{-1})_n \) and \( y = (-1 + d^{-1})_n \). The decoding procedure is described in Algorithm \( \text{[ ]} \)
Algorithm 1

1: $b_{prev} \leftarrow 0$
2: for $t = 0, 1, \ldots, x$ do
3:     $s_1 \leftarrow (-d(t + 1) - 2)_n$
4:     $s_2 \leftarrow \langle s_1 + j \rangle_n$
5:     if $(s_1 \notin \{i, j, n - 1\})$ then
6:         $L_{\Delta}(v_{s_1}, v_i) \leftarrow D_{s_2} + b_{prev}$
7:         $L_{\Delta}(v_{s_1}, v_i) \leftarrow \hat{S}_{s_1} + L_{\Delta}(v_{s_1}, v_j)$
8:         $b_{prev} \leftarrow L_{\Delta}(v_{s_1}, v_i)$
9:     end if
10:    if $(s_1 = j)$ then
11:       $L_{\Delta}(v_j, v_j) \leftarrow \hat{D}_{s_2} + b_{prev}$
12:       $L_{\Delta}(v_{j}, v_i) \leftarrow \hat{S}_{n - 2} + L_{\Delta}(v_{j}, v_j)$
13:       $b_{prev} \leftarrow L_{\Delta}(v_{j}, v_i)$
14:    end if
15:    if $s_1 = n - 1$ then
16:       $L_{\Delta}(v_{n - 1}, v_j) \leftarrow \hat{D}_{s_2} + b_{prev}$
17:    end if
18: end for
19: $b_{prev} \leftarrow 0$
20: for $t = 0, 1, \ldots, y$ do
21:     $s_1 \leftarrow (d(t + 1) - 2)_n$
22:     $s_2 \leftarrow \langle s_1 + i \rangle_n$
23:     if $(s_1 \notin \{i, j, n - 1\})$ then
24:         $L_{\Delta}(v_{s_1}, v_i) \leftarrow D_{s_2} + b_{prev}$
25:         $L_{\Delta}(v_{s_1}, v_i) \leftarrow \hat{S}_{s_1} + L_{\Delta}(v_{s_1}, v_i)$
26:         $b_{prev} \leftarrow L_{\Delta}(v_{s_1}, v_i)$
27:     end if
28:     if $(s_1 = i)$ then
29:         $L_{\Delta}(v_i, v_i) \leftarrow \hat{D}_{s_2} + b_{prev}$
30:         $L_{\Delta}(v_i, v_j) \leftarrow \hat{S}_{n - 2} + L_{\Delta}(v_i, v_j)$
31:         $b_{prev} \leftarrow L_{\Delta}(v_i, v_j)$
32:     end if
33:     if $s_1 = n - 1$ then
34:         $L_{\Delta}(v_{n - 1}, v_i) \leftarrow \hat{D}_{s_2} + b_{prev}$
35:     end if
36: end for

In order to prove the correctness of the algorithm, we define an auxiliary parameter which describes the set of uncorrected edges at the beginning of each iteration of the algorithm. More specifically, denote $\hat{F}^{(t)}$ as the set of uncorrected edges in the first loop and $\tilde{F}^{(t)}$ in the second loop, so $\hat{F}^{(0)} = F_1 \cup F_2$ and $\tilde{F}^{(0)} = F(x)$. Denote by $s_1^{(t)}, s_2^{(t)}$ the values of $s_1, s_2$, respectively on iteration $t$ in the first loop. The values of $\hat{s}_1^{(t)}$ and $\tilde{s}_2^{(t)}$ will be defined similarly for the second loop.

The values of $s_1^{(t)}$ and $s_2^{(t)}$ are given by:

$$s_1^{(t)} = \langle -d(t + 1) - 2 \rangle_n, s_2^{(t)} = \langle s_1^{(t)} + j \rangle_n = \langle -dt + i - 2 \rangle_n.$$  

Similar expressions can be derived for $\hat{s}_1^{(t)}$ and $\tilde{s}_2^{(t)}$. We also denote the following sets,

$$A = \{s_1^{(t)} | 0 \leq t_1 \leq x\}, B = \{\hat{s}_1^{(t)} | 0 \leq t_2 \leq y\}.$$  

Before proving the correctness of the algorithm, we show some more useful properties which are deferred to Appendix C.

**Claim 3.** The following properties hold:

(a) $x \neq y$ and $x + y = n - 2$.
(b) $s_1^{(x)} = \hat{s}_1^{(y)} = n - 1$.
(c) $i, j \in A$ or $i, j \in B$ but not in both.
(d) $n - 2 \notin A \cup B$.
(e) $A \cap B \{n - 1\}$.
(f) \(|A| = x + 1\) and \(|B| = y + 1\).

(g) \(s_1^{(0)} \neq i\) and \(s_1^{(0)} \neq j\).

(h) For \(0 < t \leq x\), \(s_1^{(t-1)} = j\) if and only if \(s_1^{(t)} = i\).

(i) For \(0 < t \leq x\), \(s_1^{(t)} \notin \{(i - 2)_n, (j - 2)_n\}\).

According to Claim \([3\, c]\), the variable \(s_1\) in Algorithm \([4]\) gets the values of \(i\) and \(j\) either in the first or in the second loop. Let us assume for the rest of the proof that this happens in the first loop, i.e., \(i, j \in A\), while the second case is proved similarly. We are now ready to show the correctness of the first loop by induction while the proof for the second loop is very similar.

**Lemma 13.** For all \(0 \leq t \leq x\), the following properties hold:

1. If \(s_1^{(t)} \notin \{i, j, n-1\}\) then \(D_{s_2^{(t)}} \cap F^{(t)} = \{\langle v_{s_1^{(0)}}, v_j \rangle\}\), \(S_{s_1^{(0)}} \cap F^{(t)} = \{\langle v_{s_1^{(0)}}, v_i \rangle, \langle v_{s_1^{(0)}}, v_j \rangle\}\), and the edges \(\langle v_{s_1^{(0)}}, v_i \rangle, \langle v_{s_1^{(0)}}, v_j \rangle\) are corrected on the \(t\)-th iteration.

2. If \(s_1^{(t)} = j\) then \(D_{s_2^{(t)}} \cap F^{(t)} = \{\langle v_j, v_j \rangle\}\), \(S_{n-2} \cap F^{(t)} = \{\langle v_i, v_i \rangle, \langle v_j, v_j \rangle\}\) and the edges \(\langle v_i, v_i \rangle, \langle v_j, v_j \rangle\) are corrected on the \(t\)-th iteration.

3. If \(s_1^{(t)} = n - 1\) then \(D_{s_2^{(t)}} \cap F^{(t)} = \{\langle v_{n-1}, v_j \rangle\}\) and the edge \(\langle v_{n-1}, v_j \rangle\) is corrected on the \(t\)-th iteration.

**Proof.** We prove this claim by induction on \(t\). First note that for all \(t \in [n]\), it holds that \(s_1^{(t)} \neq n - 2\) as it was proved in Claim \([3\, d]\).

**Base:** For \(t = 0\) we have \(s_1^{(0)} = \langle -d - 2 \rangle_n\), \(s_2^{(0)} = \langle i - 2 \rangle_n\), and \(F^{(0)} = F_i \cup F_j\). By Claim \([3\, g]\), \(s_1^{(0)} \neq i\). We first prove the case where \(s_1^{(0)} \notin \{j, n - 1\}\). Hence, we need to show that,

1. \(D_{\langle -d - 2 \rangle_n} \cap (F_i \cup F_j) = \{\langle v_{\langle -d - 2 \rangle_n}, v_j \rangle\}\),

2. \(S_{\langle -d - 2 \rangle_n} \cap (F_i \cup F_j) = \{\langle v_{\langle -d - 2 \rangle_n}, v_i \rangle, \langle v_{\langle -d - 2 \rangle_n}, v_j \rangle\}\).

3. The edges \(\langle v_{\langle -d - 2 \rangle_n}, v_j \rangle\) and \(\langle v_{\langle -d - 2 \rangle_n}, v_i \rangle\) are corrected on this iteration.

The proof consists of the following observations:

- According to Claim \([2\, d]\) we deduce that

\[
D_{\langle -d - 2 \rangle_n} \cap (F_i \cup F_j) = \{\langle v_{\langle i - j - 2 \rangle_n}, v_j \rangle\} = \{\langle v_{\langle -d - 2 \rangle_n}, v_j \rangle\}
\]

and therefore the edge \(\langle v_{\langle -d - 2 \rangle_n}, v_j \rangle\) is corrected in Step \([6]\) according to the constraint \(D_{\langle -d - 2 \rangle_n}\), therefore, \(L_U(v_{\langle -d - 2 \rangle_n}, v_j) = \hat{D}_{\langle -d - 2 \rangle_n}\).
• According to Claim \ref{claim:correctness_iterations} we get
  \[ S_{(-d-2)n} \cap (F_i \cup F_j) = \{ \langle v_{(-d-2)n}, v_i \rangle, \langle v_{(-d-2)n}, v_j \rangle \}, \]
  and therefore the edge \( \langle v_{(-d-2)n}, v_i \rangle \) is corrected in Step \ref{step:correctness_iterations} according to the constraint \( S_{(-d-2)n} \), by
  \[ L_d(v_{(-d-2)n}, v_i) = \tilde{S}_{(-d-2)n} + L_d(v_{(-d-2)n}, v_j). \]
  Notice that Steps \ref{step:correctness_iterations} and \ref{step:correctness_iterations} are identical. Therefore, if \( s_1^{(0)} \) equals to \( j, n - 1 \), the edge \( \langle v_j, v_j \rangle, \langle v_{n-1}, v_j \rangle \) is corrected in Step \ref{step:correctness_iterations} \ref{step:correctness_iterations} according to the constraint \( D_{(i-2)n} \), respectively. If \( s_1^{(0)} = j \), according to Claim \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} we get
  \[ S_{n-2} \cap (F_i \cup F_j) = \{ \langle v_i, v_i \rangle, \langle v_j, v_j \rangle \}, \]
  and therefore the edge \( \langle v_i, v_i \rangle \) is corrected in Step \ref{step:correctness_iterations} \ref{step:correctness_iterations} according to the constraint \( S_{n-2} \), by
  \[ L_d(v_i, v_i) = \tilde{S}_{n-2} + L_d(v_j, v_j). \]

**Step:** Assume that the induction assumption holds for \( t - 1 \), where \( t \leq x \) and we prove its correctness for \( t \). In this case, by Claim \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} and Claim \ref{claim:correctness_iterations} \ref{claim:correctness_iterations}, only \( s_1^{(x)} \) in \( A \) is equal to \( n - 1 \), so we have that \( s_1^{(t-1)} \neq n - 1 \). If \( s_1^{(t-1)} \notin \{i, j\} \), we assume that the edges \( \langle v_{s_1^{(t-1)}}, v_j \rangle \) and \( \langle v_{s_1^{(t-1)}}, v_i \rangle \) were corrected on the \( t-1 \) iteration. If \( s_1^{(t-1)} = j \), by Claim \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} it holds if and only if \( s_1^{(t)} = i \). Notice that the algorithm do nothing on this iteration and it can be assumed that the edges \( \langle v_j, v_j \rangle \) and \( \langle v_i, v_i \rangle \) were corrected on the \( t - 1 \) iteration. Therefore, we left to analyze the case where \( s_1^{(t-1)} \notin \{j, n - 1\} \). We consider the following cases:

1. \( s_1^{(t)} \notin \{j, n - 1\} \): By Claim \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} \ref{claim:correctness_iterations} we deduce that
   \[ D_{s_2^{(t)}} \cap (F_i \cup F_j) = \{ \langle v_{s_2^{(t)}}, v_i \rangle, \langle v_{s_2^{(t)}}, v_j \rangle \} = \{ \langle v_{s_1^{(t-1)}}, v_i \rangle, \langle v_{s_1^{(t)}, v_j} \rangle \}. \]
   By the induction assumption \( \langle v_{s_1^{(t-1)}}, v_i \rangle \) was corrected, so,
   \[ D_{s_2^{(t)}} \cap (F_i^{(t)} \cup F_j^{(t)}) = \{ \langle v_{s_1^{(t)}}, v_j \rangle \}, \]
   and the edge \( \langle v_{s_1^{(t)}}, v_j \rangle \) is successfully corrected in Step \ref{step:correctness_iterations} by \( D_{s_2^{(t)}} \) constraint. Furthermore, since \( s_1^{(t)} \neq n - 1 \), by Claim \ref{claim:correctness_iterations} \ref{claim:correctness_iterations},
   \[ S_{s_1^{(t)}} \cap (F_i \cup F_j) = \{ \langle v_{s_1^{(t)}}, v_i \rangle, \langle v_{s_1^{(t)}}, v_j \rangle \}, \]
   39
so it holds that $S_{s_1^{(t)}} \cap (F_{i}^{(t)} \cup F_{j}^{(t)}) = \{v_{s_1^{(t)}}, v_{i}, v_{s_1^{(t)}}, v_{j}\}$ and therefore the edge $\langle v_{s_1^{(t)}}, v_{i} \rangle$ can be successfully corrected in Step 7 by constraint $S_{s_1^{(t)}}$ and the value of $L_{\mathcal{U}}^n(v_{s_1^{(t)}}, v_{j})$. Notice that in this case $s_1^{(t-1)}$ can also be equal to $i$.

2. $s_1^{(t)} = j$ or $s_1^{(t)} = n - 1$: Since Steps 6, 11 and 16 are identical, we first correct the edge $\langle v_{s_1^{(t)}}, v_{j} \rangle$ by the $D_{s_2^{(t)}}$ constraint. In case that $s_1^{(t)} = j$, by Claim 2(d), $S_{n-2} \cap (F_{i} \cup F_{j}) = \{v_{i}, v_{j}, \{v_{j}^1, v_{j}\}\}$ so it holds that $S_{n-2} \cap (F_{i}^{(t)} \cup F_{j}^{(t)}) = \{v_{i}, v_{j}, \{v_{j}^1, v_{j}\}\}$. Therefore the edge $\langle v_{i}, v_{i} \rangle$ is corrected in Step 12 by the $S_{n-2}$ constraint and the value of $L_{\mathcal{U}}^n(v_{j}, v_{j})$.

$$\square$$

A similar lemma for the second loop is stated as follows. We omit its proof since it is very similar to the one of Lemma 13.

**Lemma 14.** For all $0 \leq t \leq y$, by assumption that $i, j \notin B$ the following properties hold:

1. If $s_1^{(t)} \neq n - 1$ then $D_{s_1^{(t)}} \cap F_{i}^{(t)} = \{v_{s_1^{(t)}}, v_{j}\}$, $S_{s_1^{(t)}} \cap F_{i}^{(t)} = \{v_{s_1^{(t)}}, v_{i}, v_{s_1^{(t)}}, v_{j}\}$, and the edges $\langle v_{s_1^{(t)}}, v_{j} \rangle, \langle v_{s_1^{(t)}}^1, v_{i} \rangle$ are corrected on the $t$-th iteration.

2. If $s_1^{(t)} = n - 1$ then $D_{s_2^{(t)}} \cap F_{i}^{(t)} = \{v_{n-1}, v_{j}\}$ and the edge $\langle v_{n-1}, v_{j} \rangle$ is corrected on the $t$-th iteration.

Let $V_1, V_2$ be the set of edges which were corrected in the first, second loop, respectively. Hence,

$$V_1 = \{v_{s_1}, v_{i}, v_{s_1}, v_{j} : s_1 \in A \setminus \{n - 1\}\},$$

$$V_2 = \{v_{s_1}, v_{j} : s_1 \notin B \setminus \{n - 1\}\}.$$ 

We also define $V = V_1 \cup V_2$, and prove the following claim in Appendix D.

**Claim 4.** The following properties hold:

(a) $V_1 \cap V_2 = \emptyset$.

(b) $|V| = 2n - 4$.

(c) $\langle v_{i}, v_{j} \rangle \notin V$.

(d) $\langle v_{n-2}, v_{i} \rangle \notin V$ and $\langle v_{n-2}, v_{j} \rangle \notin V$.

Lastly, at the end of the algorithm the following property on the set of uncorrected edges holds.
\textbf{Claim 5.} At the end of the algorithm,
\[ \bar{F}^{(y)} = \{ (v_i, v_j), (v_{n-2}, v_i), (v_{n-2}, v_j) \} \]

\textit{Proof.} Since the number of erased edges is \( 2n - 1 \), we get that
\[ |\bar{F}^{(y)}| = (2n - 1) - |V| = (2n - 1) - (2n - 4) = 3. \]

By Claim \[4\] and Claim \[4\], the edges that were not decoded yet are
\[ \{ (v_i, v_j), (v_{n-2}, v_i), (v_{n-2}, v_j) \}, \]
and therefore,
\[ \bar{F}^{(y)} = \{ (v_i, v_j), (v_{n-2}, v_i), (v_{n-2}, v_j) \}. \]

According to Claim \[2\], \( D_{(i+j)n} \cap (F_i \cup F_j) = \{ (v_i, v_j) \} \) and therefore the edge \( (v_i, v_j) \) can be reconstructed by the constraint \( D_{(i+j)n} \), that is, \( L_d(v_i, v_j) = \hat{D}_{(i+j)n} \). Since the only uncorrected edges of nodes \( v_i \) and \( v_j \) are \( (v_i, v_{n-2}), (v_j, v_{n-2}) \), they are corrected by the constraints \( S_i \) and \( S_j \). The number of constraints of this code is \( 2n - 1 \), which meets the bound in (3.4) so it is an optimal code. As mentioned before, each optimal code is also a systematic code, and thus this code is systematic.

Note that for the systematic version of the code \( C_{Ud} \), where the last two nodes are the redundancy nodes, we sometimes refer the first node among them by the \textit{single parity node} and the second one by the \textit{diagonal parity node}. The decoding algorithm presented in the proof of Theorem \[12\] is demonstrated in the next example.

\textbf{Example 4.} In this example we show a decoding scheme of \( C_{Ud} \) code \( SU-[11,9] \). We consider the case where the failed nodes are \( v_3 \) and \( v_5 \), that is, \( i = 3, j = 5 \). Therefore \( d = 2 \) and \( x = 4, y = 5 \). The undirected graph is represented by a lower-triangle-adjacency matrix. The yellow cells of the matrix represent the erased edges. As mentioned before, the decoding procedure in Algorithm \[1\] corrects all the erased edges using two loops. The first loop is represented by Steps 1-14 and the second loop is represented by Steps 15-28. In each iteration of both loops we calculate \( s_1 \) and \( s_2 \) that we show in the tables of Figure \[3\]. Thus, we get the sets \( A \) and \( B \), that is, \( A = \{ 7, 5, 3, 1, 10 \} \) and \( B = \{ 0, 2, 4, 6, 8, 10 \} \). Therefore, the first loop starts with the edge \( (v_7, v_5) \), since \( s_1^{(0)} = (d-2)_{11} = (7 - 2)_{11} = 7 \) and ends with the edge \( (v_{10}, v_5) \), since \( s_1^{(4)} = (d(x+1) - 2)_{11} = (2 \cdot 5 - 2)_{11} = 10 \). Similarly, the second loop starts with the edge \( (v_3, v_3) \) and ends with the edge \( (v_{10}, v_3) \). Notice that \( 5, 3 \in A \), so the first loop corrects the self loop edges \( (v_5, v_5) \) and \( (v_3, v_3) \). At the end of this algorithm the edge \( (v_5, v_3), (v_9, v_3), (v_9, v_5) \) that is marked with gray, is corrected with the constraint \( D_8, S_3, S_5 \), respectively.
We finish this section by showing another construction of systematic binary undirected double-node-erasure-correcting code $SU-[n,n-2]$ which is very similar to the code from Construction 6. Here, we present this construction by showing the constraints of graphs on their upper-triangle-adjacency matrices. We have this construction since it will be used in Section 3.6 for the construction of optimal binary directed double-node-erasure-correcting codes $SD-[n,n-2]$. This construction is almost a symmetric reflection of Construction 6 with respect to the main diagonal. However, we had to introduce only a single modification in which we changed between the roles of the redundancy nodes $v_{n-2}$ and $v_{n-1}$.

Let $G = (V_n, L)$ be a graph with $n$ vertices. Let us define for $h \in [n-1]$

$$S'_h = \begin{cases} \{ (\nu_h, \nu_{\ell}) \mid \ell \in [n] \setminus \{n-2\} \} , & h \in [n-2], \\ \{ (\nu_{\ell}, \nu_{\ell}) \mid \ell \in [n] \setminus \{n-2\} \} , & h = n-2, \end{cases}$$

(3.8)

and for $m \in [n]$,

$$D'_m = \{(\nu_k, \nu_{\ell}) | k, \ell \in [n-1], (k + \ell)_{n} = m \} \cup \{(v_{n-2}, v_{n-1})\}. \quad (3.9)$$

As before, the sets $S'_h$ for $h \in [n-1]$ and $D'_m$ for $m \in [n]$, will be used to represent parity constraints on the upper-triangle-adjacency matrix of each graph.

**Example 5.** The sets $S'_h, D'_m$ for $n = 7$ are marked in Fig. 5 Entries on lines with the same color belong to the same parity constraint.

Our second construction of optimal binary undirected double-node-erasure-correcting codes $SU-[n,n-2]$ works as follows.

**Construction 7.** For all $n \geq 5$ prime number let $C_{U_6}$ be the following code over graphs,

$$C_{U_6} = \left\{ G = (V_n, L) \left| \begin{array}{l} (a) \sum_{(\nu_i, \nu_j) \in S'_h} L_{U}(\nu_i, \nu_j) = 0, h \in [n-1] \\ (b) \sum_{(\nu_i, \nu_j) \in D'_m} L_{U}(\nu_i, \nu_j) = 0, m \in [n] \end{array} \right. \right\}. \quad (3.9)$$

42

Figure 3.4: A decoding scheme of an optimal binary undirected double-node-erasure-correcting code $SU-[11,9]$. 

(a) Simulation of the algorithm

(b) Corrected edge order
Figure 3.5: The neighborhood and the diagonal sets.

**Theorem 15.** The code $\mathcal{C}_U^6$ is an optimal binary undirected double-node-erasure-correcting code.

We will not prove here the correctness of the code $\mathcal{C}_U^6$ since its construction is very similar to one of the code $\mathcal{C}_U^5$. However, note that when constructing the code $\mathcal{C}_U^6$, we switched the roles of the last two redundancy nodes such that the first node is the diagonal parity node and the second node is the single parity node. We still present here a decoding algorithm of this code for the more challenging case when the failed nodes are $v_i, v_j$ and $i, j \in [n - 2]$. Its correctness is similar to the one of Algorithm 1 and is thus omitted.

First, we denote the single parity syndromes for $h \in [n - 1] \setminus \{i, j\}$ by

$$\tilde{S}_h = \sum_{(v_k, v_\ell) \in S_h \setminus (F_i \cup F_j)} L_{U}(v_k, v_\ell),$$

and the diagonal parity syndromes for $m \in [n]$ by

$$\tilde{D}'_m = \sum_{(v_k, v_\ell) \in D'_m \setminus (F_i \cup F_j)} L_{U}(v_k, v_\ell).$$

Let $x' = (-1 + d^{-1})_n$ and $y' = (-1 - d^{-1})_n$. The decoding procedure for this case is described in Algorithm 2.

The decoding algorithm presented in the proof of Theorem 15 is demonstrated in the next example.

**Example 6.** In this example we show a decoding scheme of the code $\mathcal{C}_U^6$ for $n = 11$. We consider the case where the failed nodes are $v_3$ and $v_5$, that is, $i = 3, j = 5$. Therefore $d = 2$ and $x = 5, y = 4$. The undirected graph is represented by an upper-triangle-adjacency matrix. The yellow cells of the matrix represent the erased edges. As mentioned before, the decoding procedure in Algorithm 2 corrects all erased edges using two loops. The first loop is represented by Steps 1-14 and the second loop is represented by Steps 15-28. In each iteration of both loops we
Algorithm 2

1: \( b_{prev} \leftarrow 0 \)
2: for \( t = 0,1,\ldots,z' \) do
3: \( s_1 \leftarrow (-d(t+1) - 1)_n \)
4: \( s_2 \leftarrow (s_1 + j)_n \)
5: if \( (s_1 \notin \{i,j,n-2\}) \) then
6: \( L_d(v_1,v_i) \leftarrow D_{s_2} + b_{prev} \)
7: \( L_d(v_1,v_i) \leftarrow S_1 + L_d(v_1,v_j) \)
8: \( b_{prev} \leftarrow L_d(v_1,v_i) \)
9: end if
10: if \( (s_1 = j) \) then
11: \( L_d(v_j,v_j) \leftarrow D_{s_2} + b_{prev} \)
12: \( L_d(v_i,v_j) \leftarrow S_1 + L_d(v_i,v_j) \)
13: \( b_{prev} \leftarrow L_d(v_i,v_j) \)
14: end if
15: if \( s_1 = n - 2 \) then
16: \( L_d(v_{n-2},v_j) \leftarrow D_{s_2} + b_{prev} \)
17: end if
18: end for
19: \( b_{prev} \leftarrow 0 \)
20: for \( t = 0,1,\ldots,y' \) do
21: \( s_1 \leftarrow (d(t+1) - 1)_n \)
22: \( s_2 \leftarrow (s_1 + i)_n \)
23: if \( (s_1 \notin \{i,j,n-2\}) \) then
24: \( L_d(v_1,v_i) \leftarrow D_{s_2} + b_{prev} \)
25: \( L_d(v_1,v_i) \leftarrow S_1 + L_d(v_1,v_i) \)
26: \( b_{prev} \leftarrow L_d(v_1,v_i) \)
27: end if
28: if \( (s_1 = i) \) then
29: \( L_d(v_i,v_i) \leftarrow D_{s_2} + b_{prev} \)
30: \( L_d(v_i,v_j) \leftarrow S_1 + L_d(v_i,v_i) \)
31: \( b_{prev} \leftarrow L_d(v_i,v_j) \)
32: end if
33: if \( s_1 = n - 2 \) then
34: \( L_d(v_{n-2},v_i) \leftarrow D_{s_2} + b_{prev} \)
35: end if
36: end for

calculate \( s_1 \) and \( s_2 \) that we show in the tables of Figure 9. Thus, we get the sets \( A \) and \( B \) to be \( A = \{8,6,4,2,0,9\} \) and \( B = \{1,3,5,7,9\} \). Therefore, the first loop starts with the edge \( \langle v_8, v_5 \rangle \), since \( s_1^{(0)} = (-d-1) = (-2-1)_{11} = 8 \) and ends with the edge \( \langle v_9, v_5 \rangle \), since \( s_1^{(5)} = (-d(x+1) - 1)_{11} = (-2\cdot 6 - 1)_{11} = 9 \). Similarly, the second loop starts with the edge \( \langle v_1, v_3 \rangle \) and ends with the edge \( \langle v_9, v_3 \rangle \). Notice that \( 3,5 \in B \), so the second loop corrects the self loop edges \( \langle v_3, v_3 \rangle \) and \( \langle v_5, v_5 \rangle \). At the end of this algorithm the edge \( \langle v_5, v_3 \rangle, \langle v_{10}, v_3 \rangle, \langle v_{10}, v_5 \rangle \) that is marked with gray, is corrected using the constraint \( D'_{s_2}, S'_{s_2} \), respectively.

![Simulation of the algorithm](image1)

![Corrected edge order](image2)

Figure 3.6: A decoding scheme of an optimal binary undirected double-node-erasure-correcting code SLU-[11, 9].

Note, that the code \( C_{U_6} \) is also an optimal and therefore it is systematic.
3.6 Optimal Binary Directed Double-Node-Erasure-Correcting Codes

In this section we combine between Constructions 6 and 7 in order to generate an optimal binary directed double-node-erasure-correcting code. The main idea here is to use Construction 6 in order to correct the backward edges \((v_i, v_j)\) for \(i > j\), i.e. the edges in the lower part of the matrix, and Construction 7 for the correction of the forward edges \((v_i, v_j)\) for \(i < j\) which are the edges in the upper part of the matrix. However, since the self loops are involved in both of these parts, we will have to carefully interleave between the two constructions. In particular, this dependency affects also the decoding of the two constructions which will have to be combined as well. Throughout this section we assume that \(n \geq 5\) is a prime number.

Let \(G = (V_n, L_D)\) be a directed graph with \(n\) vertices and let \(G_2 = (V_n, L_U)\) be an undirected graph. We use the same definitions of the sets \(S_h, S'_h, D_m, D'_m\), \(h \in [n-2], m \in [n]\) from (3.8), (3.9), respectively, and let \(F_t, t \in [n]\) be a failure set.

For \(i, j \in [n]\), not necessarily distinct, let \(\langle v_i, v_j \rangle^\downarrow\) be the edge directed from \(v_{\max\{i,j\}}\) to \(v_{\min\{i,j\}}\), i.e., \(\langle v_i, v_j \rangle^\downarrow = (v_{\max\{i,j\}}, v_{\min\{i,j\}})\), and similarly \(\langle v_i, v_j \rangle^\uparrow = (v_{\min\{i,j\}}, v_{\max\{i,j\}})\) is the edge directed from \(v_{\min\{i,j\}}\) to \(v_{\max\{i,j\}}\). Next we define the constraint sets for \(G\).

For \(h \in [n-2]\) the neighborhood-edge sets \(S_h^\downarrow, S_h^\uparrow\) are defined by

\[
S_h^\downarrow = \{ \langle v_i, v_j \rangle^\downarrow | \langle v_i, v_j \rangle \in S_h \},
\]

\[
S_h^\uparrow = \{ \langle v_i, v_j \rangle^\uparrow | \langle v_i, v_j \rangle \in S'_h \}.
\]

Furthermore, for \(m \in [n]\) the diagonal-edge sets \(D_m^\downarrow, D_m^\uparrow\) are defined by

\[
D_m^\downarrow = \{ \langle v_i, v_j \rangle^\downarrow | \langle v_i, v_j \rangle \in D_m \},
\]

\[
D_m^\uparrow = \{ \langle v_i, v_j \rangle^\uparrow | \langle v_i, v_j \rangle \in D'_m \}.
\]

and lastly for \(t \in [n]\) the failure-edge sets \(F_t^\downarrow, F_t^\uparrow\) are defined by

\[
F_t^\downarrow = \{ \langle v_i, v_j \rangle^\downarrow | \langle v_i, v_j \rangle \in F_t \},
\]

\[
F_t^\uparrow = \{ \langle v_i, v_j \rangle^\uparrow | \langle v_i, v_j \rangle \in F_t \}.
\]

Example 7. The sets \(S_h^\downarrow, S_h^\uparrow, D_m^\downarrow, D_m^\uparrow\) for \(n = 7\) are marked in Fig. 7. Entries on lines with the same color belong to the same parity constraint.

The following claim for the directed case is very similar to the corresponding one from Claim 4. Thus, we omit its proof.

Claim 6.

(a) For all distinct \(h, i \in [n-2]\), \(S_h^\downarrow \cap F_i^\uparrow = \{ \langle v_h, v_i \rangle^\uparrow \} \).
(b) For all $i \in [n-2]$, $D_{(i-2)n}^\dagger \cap (F_i^\dagger \cup F_{n-2}^\dagger) = \{(v_{n-1}, v_{n-2})\}$.

(c) For all $i \in [n-2]$, $s \in [n] \setminus \{(i-2)n\}$, $D_s^\dagger \cap F_i^\dagger = \{(v_{(s-i)n}, v_i)^\dagger\}$.

(d) For all distinct $i, j \in [n-2]$, $D_{(i+j)n}^\dagger \cap F_j^\dagger = \{(v_j, v_i)\}$ and $D_{(i+j)n}^\dagger \cap F_j^\dagger = \{(v_i, v_j)\}$.

We are now ready to present the construction of optimal binary directed double-node-erasure-correcting codes.

**Construction 8.** For all $n \geq 5$ prime number let $C_{D_n}$ be the following code.

$$C_{D_n} = \left\{ \begin{array}{l}
(a) \sum_{(v_i, v_j) \in S_i^\dagger} L_D(v_i, v_j) = 0, h \in [n-2] \\
(b) \sum_{(v_i, v_j) \in D_{h}^\dagger} L_D(v_i, v_j) = 0, m \in [n] \\
(c) \sum_{(v_i, v_j) \in S_i^\dagger} L_D(v_i, v_j) = 0, h \in [n-2] \\
(d) \sum_{(v_i, v_j) \in D_{n-3}^\dagger} L_D(v_i, v_j) = 0, m \in [n] 
\end{array} \right\}.$$

In this binary construction we did not use the constraints that were derived from the two sets $S_{n-2}^\dagger$ and $S_{n-2}^\dagger$ (i.e., the constraints on the main diagonal). We will show that an encoding scheme of this code is also systematic. Assume the first $n-2$ nodes carry the information symbols on their edges. It can be verified that for all $i \in [n-2]$ the encoding with each of the constraints $S_i^\dagger, S_i^\dagger, D_{n-3}^\dagger$ and $D_{n-3}^\dagger$ is possible since each of them encodes its appropriate redundancy edge using a simple parity code. For all $m \in [n] \setminus \{n-3\}$ the encoding with constraints $D_{n}^\dagger$ and $D_{n}^\dagger$ is also possible since the edges $(v_{n-1}, v_{n-2})$ and $(v_{n-2}, v_{n-1})$ were encoded with respect to $D_{n-3}^\dagger$ and $D_{n-3}^\dagger$. Hence, in the next proof for the correctness of the construction we will refer to it as a systematic construction.

**Theorem 16.** The code $C_{D_n}$ is an optimal binary directed double-node-erasure-correcting code.
Proof. Assume that the nodes \( i, j \in [n], i < j \) are failed. We will show the correctness of this construction by splitting it into three cases.

**Case 1:** \( i = n - 2, j = n - 1 \). In this case the information edges have not been erased, therefore, we can use the encoding rules of \( C_{\mathcal{D}_r} \) to correct the failed redundancy nodes.

**Case 2:** \( i \in [n - 2], j = n - 2 \). In this case, by Claim 6(a) for \( h \in [n - 2] \setminus \{i\} \), \( S_h^\uparrow \cap F_i^\uparrow = \{\langle v_h, v_i\rangle^\uparrow\} \) so the following \( n - 3 \) edges connected to the \( i \)-th node can be corrected by constraint (c) of Construction 8, that is,

\[
L_D(\langle v_h, v_i\rangle^\uparrow) = \sum_{(v_k, v_l) \in S_h^\uparrow \setminus \{(v_h, v_i)\}} L_D(v_k, v_l).
\]

Notice that we corrected \( n - 3 \) information edges on the upper-triangle-adjacency matrix of the graph, where the edges \((v_i, v_i), (v_i, v_{n-2})\) and \((v_i, v_{n-1})\) are not corrected yet. From Claim 6(b) we get that \( D_{(i-2)}^\uparrow \cap (F_i^\downarrow \cup F_{n-2}^\downarrow) = \{(v_{n-1}, v_{n-2})\} \). Therefore the edge \((v_{n-1}, v_{n-2})\) can be corrected using constraint (b) of Construction 8.

\[
L_D(v_{n-1}, v_{n-2}) = \sum_{(v_k, v_l) \in D_{(i-2)}^\uparrow \setminus \{(v_{n-1}, v_{n-2})\}} L_D(v_k, v_l).
\]

By Claim 6(c) for all \( m \in [n] \setminus \{i - 2\} \), \( D_m^\uparrow \cap F_i^\downarrow = \{\langle v_{(m-i)}\rangle_n, v_i\rangle^\downarrow\} \). Since \((v_{n-1}, v_{n-2})\) is the only edge that intersects between constraints \( D_m^\uparrow \) and \( F_{n-2}^\downarrow \), we get that \( D_m^\uparrow \cap (F_i^\downarrow \cup F_{n-2}^\downarrow) = \{(v_{(m-i)}\rangle_n, v_i\rangle^\downarrow, (v_{n-1}, v_{n-2})\} \). Therefore, the following edges connected to the node \( v_i \) are decoded by constraint (b) of Construction 8.

\[
L_D(\langle v_{(m-i)}\rangle_n, v_i\rangle^\downarrow) = \sum_{(v_k, v_l) \in D_m^\uparrow \setminus \{(v_{(m-i)}\rangle_n, v_i\rangle^\downarrow\}} L_D(v_k, v_l).
\]

It can be verified that we corrected another \( n \) edges on the lower-triangle-adjacency matrix of the graph, where there are no uncorrected information edges left. Thus, the remaining edges can be successfully decoded according to the encoding rules of the code. For \( j = n - 1 \) the proof is very similar and thus we omit its details.

**Case 3:** The neighborhood syndromes \( \hat{S}_h^\downarrow, \hat{S}_h^\uparrow \) are defined by

\[
\hat{S}_h^\downarrow = \sum_{(v_k, v_l) \in S_h^\downarrow \setminus (F_i^\downarrow \cup F_j^\downarrow)} L_D(v_k, v_l),
\]

\[
\hat{S}_h^\uparrow = \sum_{(v_k, v_l) \in S_h^\uparrow \setminus (F_i^\downarrow \cup F_j^\downarrow)} L_D(v_k, v_l).
\]
Algorithm 3

<table>
<thead>
<tr>
<th>Loop I</th>
<th>Loop II</th>
<th>Loop III</th>
<th>Loop IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: ( b_{prev} \leftarrow 0 )</td>
<td>15: ( b_{prev} \leftarrow 0 )</td>
<td>29: ( b_{prev} \leftarrow 0 )</td>
<td>43: ( b_{prev} \leftarrow 0 )</td>
</tr>
<tr>
<td>2: for ( i = 0, 1, \ldots, y ) do</td>
<td>16: for ( i = 0, 1, \ldots, y ) do</td>
<td>30: for ( i = 0, 1, \ldots, y' ) do</td>
<td>44: for ( i = 0, 1, \ldots, y' ) do</td>
</tr>
<tr>
<td>3: ( s_i = (s_{i+1} + 2)_n )</td>
<td>17: ( s_i = (s_{i+1} + 2)_n )</td>
<td>31: ( s_i = (s_{i+1} + 2)_n )</td>
<td>45: ( s_i = (s_{i+1} + 2)_n )</td>
</tr>
<tr>
<td>4: ( s_i = (s_{i+1} + 2)_n )</td>
<td>18: ( s_i = (s_{i+1} + 2)_n )</td>
<td>32: ( s_i = (s_{i+1} + 2)_n )</td>
<td>46: ( s_i = (s_{i+1} + 2)_n )</td>
</tr>
<tr>
<td>5: if ( (s_j, s_{j+1}) ) then</td>
<td>19: if ( (s_j, s_{j+1}) ) then</td>
<td>33: if ( (s_j, s_{j+1}) ) then</td>
<td>47: if ( (s_j, s_{j+1}) ) then</td>
</tr>
<tr>
<td>6: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>20: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>34: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>48: ( L_D(v_i, v_j) = b_{prev} + D )</td>
</tr>
<tr>
<td>7: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>21: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>35: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>49: ( L_D(v_i, v_j) = b_{prev} + D )</td>
</tr>
<tr>
<td>8: ( b_{prev} = L_D(v_i, v_j) )</td>
<td>22: ( b_{prev} = L_D(v_i, v_j) )</td>
<td>36: ( b_{prev} = L_D(v_i, v_j) )</td>
<td>50: ( b_{prev} = L_D(v_i, v_j) )</td>
</tr>
<tr>
<td>9: if ( (s_j, s_{j+1}) ) then</td>
<td>23: if ( (s_j, s_{j+1}) ) then</td>
<td>37: if ( (s_j, s_{j+1}) ) then</td>
<td>51: if ( (s_j, s_{j+1}) ) then</td>
</tr>
<tr>
<td>10: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>24: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>38: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>52: ( L_D(v_i, v_j) = b_{prev} + D )</td>
</tr>
<tr>
<td>11: Wait until ((s_i, v_j)) is corrected.</td>
<td>25: Wait until ((s_i, v_j)) is corrected.</td>
<td>39: Wait until ((s_i, v_j)) is corrected.</td>
<td>53: Wait until ((s_i, v_j)) is corrected.</td>
</tr>
<tr>
<td>12: ( b_{prev} = L_D(v_i, v_j) )</td>
<td>26: ( b_{prev} = L_D(v_i, v_j) )</td>
<td>40: ( b_{prev} = L_D(v_i, v_j) )</td>
<td>54: ( b_{prev} = L_D(v_i, v_j) )</td>
</tr>
<tr>
<td>13: if ( s_i = n - 1 ) then</td>
<td>27: if ( s_i = n - 1 ) then</td>
<td>41: if ( s_i = n - 2 ) then</td>
<td>55: if ( s_i = n - 2 ) then</td>
</tr>
<tr>
<td>14: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>28: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>42: ( L_D(v_i, v_j) = b_{prev} + D )</td>
<td>56: ( L_D(v_i, v_j) = b_{prev} + D )</td>
</tr>
</tbody>
</table>

and the diagonal syndromes \( \hat{D}_m^\downarrow, \hat{D}_m^\uparrow \) are defined by

\[
\hat{D}_m^\downarrow = \sum_{(v_k, v_{i'}) \in \mathcal{D}_m \setminus (\mathcal{F}_i^\downarrow \cup \mathcal{F}_i^\uparrow)} L_D(v_k, v_{i'})
\]

\[
\hat{D}_m^\uparrow = \sum_{(v_k, v_{i'}) \in \mathcal{D}_m \setminus (\mathcal{F}_i^\uparrow \cup \mathcal{F}_i^\downarrow)} L_D(v_k, v_{i'}).
\]

Let \( d = \langle j - i \rangle_n, x = \langle -1 - d^{-1} \rangle_n, y = \langle -1 + d^{-1} \rangle_n, x' = \langle -1 + d^{-1} \rangle_n \) and \( y' = \langle -1 - d^{-1} \rangle_n \). The decoding procedure for the code \( \mathcal{C}_{\mathcal{D}_7} \) in this case is described in Algorithm 3.

This algorithm consists of four loops marked as Loop I, II, III, and IV.

For \( Y \in \{ I, II, III, IV \} \), denote by \( s^{(t)}_{1,Y} \) the value of the variable \( s_1 \) on iteration \( t \) of Loop \( Y \). These values of \( s^{(t)}_{1,Y} \) are given by:

\[
s^{(t)}_{1,I} = \langle -d(t + 1) - 2 \rangle_n, s^{(t)}_{1,II} = \langle d(t + 1) - 2 \rangle_n, s^{(t)}_{1,III} = \langle -d(t + 1) - 1 \rangle_n, s^{(t)}_{1,IV} = \langle d(t + 1) - 1 \rangle_n.
\]

Next, we denote the following four sets:

\[
A = \{ s^{(t)}_{1,I} : t \in [x + 1] \}, B = \{ s^{(t)}_{1,II} : t \in [y + 1] \},
\]

\[
A' = \{ s^{(t')}_{1,III} : t' \in [x' + 1] \}, B' = \{ s^{(t')}_{1,IV} : t' \in [y' + 1] \}.
\]

**Claim 7.** The indices \( i, j \) satisfy the following property: \( i, j \in A \cap B' \) or \( i, j \in A' \cap B \), but not in both.

**Proof.** Notice that sets \( A \) and \( B \) are defined similarly to (3.7). In Claim 3.10, it was stated that \( i, j \in A \) or \( i, j \in B \), but not in both, and it is possible to show that the same property holds for \( A' \) and \( B' \). Without loss of generality, let us assume that \( i, j \in A \). Let \( 0 \leq t < x - 1 \) be a step in which \( s^{(t)}_{1,I} = j \) and therefore

48
Therefore we can calculate $s_{1,1}^{(x-(t+1))}$, hence $s_{1,1}^{(x-(t+1))} - j = s_{1,1}^{(x-(t+1))} - s_{1,1}^{(t)} =\langle d(x - (t + 1) + 1) - 1 \rangle_n - \langle -d(t + 1) - 2 \rangle_n = \langle d(x - t) + d(t + 1) + 1 \rangle_n = \langle d(x + 1) \rangle_n = \langle d((-1 - d - 1) + 1) \rangle_n = -1 + 1 = 0,$

and therefore $s_{1,1}^{(x-(t+1))} = j$. By definition of $s_{1,1}^{(t)}$, we can see that

$$s_{1,1}^{(x-(t+2))} = s_{1,1}^{(x-(t+1))} - d = i,$$

where $t+2 \leq x$ and therefore $i, j \in B'$. The opposite direction is proved similarly.

The decoding Algorithm 3 for this case combines Algorithm 1 and Algorithm 2, where Algorithm 1 is used to decode the lower-triangle-adjacency matrix and Algorithm 2 is used to decode the upper-triangle-adjacency matrix. However, since we did not use the constraints of the two sets $S_{n-2}$ and $S_{n-2}'$ on the main diagonal, we had to replace Step 11, 25 in Algorithm 1, Algorithm 2 with the command wait until $(v_i, v_j)$ is corrected, wait until $(v_j, v_i)$ is corrected, respectively. According to Claim 7, the indices $i, j$ satisfy $i, j \in A \cap B'$ or $i, j \in A' \cap B$ but not both. Without loss of generality, assume that $i, j \in A \cap B'$. Therefore, in this case, Loops II and III of Algorithm 3 will not be affected by the main diagonal constraint. This holds since the edges $(v_i, v_i)$ and $(v_j, v_j)$ are not corrected in these two loops as the conditions in Steps 23 and 37 will not hold. Hence, these two loops operate and succeed exactly as done in Algorithm 1 and Algorithm 2. This does not hold for Loops I and IV. Namely, Loop I, IV operates exactly as Algorithm 1, Algorithm 2 until Loop I, IV reaches Step 11, 53, respectively. Here we notice that according to Algorithm 1 in Step 11, the algorithm was supposed to correct the edge $(v_i, v_i)$ according to the constraint on the main diagonal. Similarly, in Step 53, the algorithm was supposed to correct the edge $(v_j, v_j)$ according to the constraint on the main diagonal. However, since the edge $(v_i, v_i)$ is corrected in Loop IV and the edge $(v_j, v_j)$ is corrected in Loop I, all we need to do in Step 11 is to wait for the edge $(v_i, v_i)$ to be corrected and in the same way in Step 53 for the edge $(v_j, v_j)$ to be corrected. Then, the rest of these two loops proceed to correct the remaining edges as done in Algorithm 1 and Algorithm 2.

Lastly, from Claim 3, $D_{(i+j)n} \cap F_j^\dagger = \{(v_j, v_i)\}$ and $D_{(i+j)n} \cap F_j^\dagger = \{(v_i, v_j)\}$, so the last two information edges $(v_j, v_i)$ and $(v_i, v_j)$ are corrected by constraints $\hat{D}_{(i+j)n}^\dagger$ and $\hat{D}_{(i+j)n}^\dagger$, respectively. Since all of the information edges were corrected, we can correct the remaining uncorrected redundancy edges $(v_{n-2}, v_i), (v_{n-2}, v_j), (v_i, v_{n-1})$ and $(v_j, v_{n-1})$ using our encoding rules.
Notice, that the number of constraints of this code is $4n - 4$, which meets the bound in (3.3) so it is an optimal code.

The decoding algorithm presented in the proof of Theorem II is demonstrated in the next example.

**Example 8.** In this example we show a decoding scheme of the code $C_{D_7}$ for $n = 11$. We consider the case where the failed nodes are $v_3$ and $v_5$, that is, $i = 3, j = 5$. Therefore $d = 2$ and $x = y = 4, x' = y' = 5$. The decoding procedure in Algorithm 3 corrects all the erased edges using four loops, where Loop I, II, III, IV is represented by Steps 1-14, 15-28, 29-42 Steps 43-56, respectively. We use here the lower-triangle-adjacency matrix for Loop I (red) and Loop II (green) and the upper-triangle-adjacency matrix for Loop III (black) and Loop IV (blue). The yellow cells of the matrix represent the erased edges. In each iteration of both loops we calculate the values of $s_{1,1}, s_{2,1}, s_{1,2}, s_{1,2}, s_{1,3}, s_{2,3}, s_{1,4}$ and $s_{2,4}$, that we show in the tables of Figure 5. Thus, we get the sets $A, B, A', B'$, where $A = \{7, 5, 3, 1, 10\}, B = \{0, 2, 4, 6, 8, 10\}, A' = \{8, 6, 4, 2, 0, 9\}$ and $B' = \{1, 3, 5, 7, 9\}$. Loop I starts with the edge $(v_7, v_5)$, and ends with the edge $(v_{10}, v_5)$ and Loop II starts with the edge $(v_3, v_9)$ and ends with the edges $(v_{10}, v_3)$. Similarly, Loop III starts with the edge $(v_5, v_8)$, and ends with the edge $(v_5, v_9)$ and finally Loop IV starts with the edge $(v_1, v_3)$ and ends with the edge $(v_3, v_9)$. Loop I, IV corrects the self loop $(v_5, v_5), (v_3, v_3)$, respectively. At the end of this algorithm, the edge $(v_5, v_3), (v_9, v_3), (v_9, v_5)$ that is marked with gray in the lower-triangle-adjacency matrix, is corrected using the constraint $D_8^5, S_3^5, S_5^5$, respectively. Similarly, the edge $(v_3, v_5), (v_3, v_{10}), (v_5, v_{10})$ that is marked with gray in the upper-triangle-adjacency matrix, is corrected using the constraint $D_8^5, S_3^5, S_5^5$, respectively.

### 3.7 Bounds on Optimal Undirected $\rho$-Node-Erasure-Correcting Codes over $\mathbb{F}_q$

In this section we study necessary conditions on the existence of optimal undirected $\rho$-node-erasure-correcting codes over $\mathbb{F}_q$ with $n$ nodes. For the special case of $\rho = n - 2$ we will show a necessary and sufficient condition and explicitly find the number of such codes.

Every linear code over undirected graphs $U-\langle n, k_U \rangle_{\mathbb{F}_q}$ can be represented by a generator matrix $G$ of dimensions $k_U \times \binom{n+1}{2}$ over $\mathbb{F}_q$. We denote the columns of the generator matrix $G$ by the indices of the set $\{(i, j) \in [n]^2 \mid i \geq j\}$, in their lexicographic order, so the column indexed by $(i, j)$ is $g_{i,j}$. For all $1 \leq \ell \leq n$, let $S_\ell$ be the set of all subsets of $[n]$ of size $\ell$, that is,

$$S_\ell = \{B \mid B \subseteq [n], |B| = \ell\}. \quad (3.10)$$

For each $B \in S_\ell$, denote the column set $V_B$ by,

$$V_B = \{g_{i,j} \mid (i, j) \in B^2, i \geq j\}. \quad (3.11)$$
Figure 3.8: A decoding scheme of an optimal binary directed double-node-erasure-correcting code $SD-[11,9]$. Clearly, the size of $V_B$, where $B \in S_\ell$, is $|V_B| = \binom{\ell+1}{2}$. The following lemma states a necessary and sufficient condition on the generator matrix of codes over undirected graphs to be optimal codes.

**Lemma 17.** Let $C_\mathcal{U}$ be a linear code over undirected graphs $\mathcal{U}=[n,k_\mathcal{U} = \binom{k+1}{2}]_{\mathbb{F}_q}$ and let $G$ be its generator matrix. Then, $C_\mathcal{U}$ is an optimal undirected $(n-k)$-node-erasure-correcting code over $\mathbb{F}_q$ if and only if for all $B \in S_k$, the columns of $V_B$ are linearly independent.

**Proof.** Let $u = (u_1, u_2, \ldots, u_{\binom{k+1}{2}}) \in \mathbb{F}_q^{\binom{k+1}{2}}$ be an information vector encoded with the matrix $G$. Assume that there was an erasure of $\rho = n - k$ nodes and let $B \in S_k$ be the set of $k$ remaining nodes. Let $c_1, c_2, \ldots, c_{\binom{k+1}{2}}$ be the information symbols on the edges of the $k$ remaining nodes in their lexicographic
order. Similarly, let \( g_1, g_2, \ldots, g_{\binom{k+1}{2}} \) be the columns of \( V_B \) in their lexicographic order. Finding the information vector \( u = (u_1, u_2, \ldots, u_{\binom{k+1}{2}}) \) is achieved by solving the following equations system

\[
[u_1, u_2, \ldots, u_{\binom{k+1}{2}}] \cdot [g_1, g_2, \ldots, g_{\binom{k+1}{2}}] = [c_1, c_2, \ldots, c_{\binom{k+1}{2}}],
\]

that has a unique solution if and only if the columns of \( V_B \) are linearly independent.

### 3.7.1 The \( \rho = n - 2 \) case

In this section we show necessary and sufficient conditions for the existence of optimal undirected \((n - 2)\)-node-erasure-correcting codes over \( \mathbb{F}_q \), and we also find the number of such codes.

**Theorem 18.** For all positive integer \( n \geq 3 \) and prime power \( q \), there exists an optimal undirected \((n - 2)\)-node-erasure-correcting code over \( \mathbb{F}_q \) if and only if

\[
q^2 + q + 2 > n,
\]

and in this case, the number of such codes is

\[
q^2(2^{\binom{n}{2}}) - (q^2 + q + 1)! \left(\frac{q^2 + q + 1}{q^2 + q + 1 - n}\right)!
\]

**Proof.** Let \( C \) be an optimal code over undirected graphs \( \mathcal{U} [n, 3]_{\mathbb{F}_q} \), and let \( G \) be its generator matrix. In this case, for \( 0 \leq i < j \leq n - 1 \) each column \((i, j)\) of \( G \) is a vector from \( \mathbb{F}_q^3 \). According to Lemma 17, for all \( i, j \in [n] \), the columns \( g_{i,i}, g_{i,j}, g_{j,j} \) are linearly independent. We will first find the number of possible columns for \( g_{i,i} \) and then for \( g_{i,j} \). For all distinct \( i, j \in [n] \), the columns \( g_{i,i}, g_{j,j} \) have to be linearly independent. First, for \( g_{0,0} \) we have \( q^3 - 1 \) options, all the vectors of \( \mathbb{F}_q^3 \) except for the zero vector. Then, for \( 1 \leq i \leq n - 1 \) we deduce that the number of valid options for \( g_{i,i} \) is

\[
(q^3 - 1) - (q - 1)i,
\]

since it cannot be a linear combination of any previous column \( g_{k,k} \) for \( 0 \leq k < i \). Hence, for \( i = n - 1 \) we require that \((q^3 - 1) - (q - 1)(n - 1) > 0\), that is,

\[
q^2 + q + 2 > n,
\]

which is a necessary condition for the existence of the code \( C \).

Next, for all \( 0 \leq i < j \leq n - 1 \) the column \( g_{i,j} \) cannot be linearly dependent on the columns \( g_{i,i} \) and \( g_{j,j} \), and therefore there are

\[
(q^3 - 1) - (q^2 - 1) = q^2(q - 1)
\]
options to choose the column $g_{i,j}$. Together, we conclude that the number of such codes will be the composition of all possible options, that is,

$$\begin{align*}
[q^2(q-1)](n) \prod_{i=0}^{n-1} [(q^2 - 1) - (q - 1)i] \\
= [q^2(q-1)](n) \prod_{i=0}^{n-1} [(q - 1)(q^2 + q + 1) - (q - 1)i] \\
= [q^2(q-1)](n) \prod_{i=0}^{n-1} [(q^2 + q + 1 - i)] \\
= [q^2(q-1)](n-1) \prod_{i=0}^{n-1} [(q^2+q+1-i)] \\
= q^2(q-1)(n+1) (q^2+q+1)! \\
\quad (q^2+q+1-n)!.
\end{align*}$$

Since for $n = q^2 + q + 1$ the number of such codes is a positive number, the condition in (3.12) is necessary and sufficient. \hfill \square

### 3.7.2 Arbitrary $\rho$

In this section we study a sufficient condition on the existence of optimal undirected $\rho$-node-erasure-correcting codes over $\mathbb{F}_q$, where $\rho = n - k$. For the rest of this section, we assume that $k$ is even, $t = k/2$, and we let $C_U$ be an optimal undirected $(n-k)$-node-erasure-correcting code $U-[n,(k+1)/2] \mathbb{F}_q$, and $G$ is its generator matrix. In order to find a necessary condition for the existence of $C_U$, we find a lower bound on the number of vectors in the set $\bigcup_{B \in S} \text{span} \ V_B$, which is then translated into an upper bound on the value of $n$, since $\bigcup_{B \in S} \text{span} \ V_B \subseteq \mathbb{F}_q^{(k+1)/2}$.

We first prove the following claim.

**Claim 8.** For all two distinct $B_1, B_2 \subseteq [n]$, such that $|B_1 \cup B_2| \leq k$, the columns of the set $V_{B_1} \cup V_{B_2}$ are linearly independent and

$$\left( \text{span} \ V_{B_1} \cap \text{span} \ V_{B_2} \right) = \text{span} \ (V_{B_1} \cap V_{B_2}).$$

**Proof.** According to Lemma 17, for all $B \in S_k$, the columns of $V_B$ are linearly independent, and this property holds for any set $B$ of size at most $k$. For all $B_1, B_2 \subseteq [n]$, such that $|B_1 \cup B_2| \leq k$, the columns of $V_{B_1 \cup B_2}$ are linearly independent and in particular also the columns of the set $V_{B_1} \cup V_{B_2}$.

To prove the second part of this claim, first note that $V_{B_1} \cap V_{B_2} \subseteq V_{B_1}$ and hence $\text{span} \ (V_{B_1} \cap V_{B_2}) \subseteq \text{span} \ V_{B_1}$ and similarly $\text{span} \ (V_{B_1} \cap V_{B_2}) \subseteq \text{span} \ V_{B_2}$, that is,

$$\text{span} \ (V_{B_1} \cap V_{B_2}) \subseteq \left( \text{span} \ V_{B_1} \cap \text{span} \ V_{B_2} \right).$$
Next, assume that $v \in \left( \text{span} V_{B_1} \cap \text{span} V_{B_2} \right)$. Denote the vector set of $V_{B_1}, V_{B_2}$ by \( \{ u_0, u_1, \ldots, u_{|V_{B_1}| - 1} \} \), \( \{ w_0, w_1, \ldots, w_{|V_{B_2}| - 1} \} \), respectively. Therefore, there are coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{|V_{B_1}| - 1}$ over $\mathbb{F}_q$ not all of them zero, and coefficients $\beta_0, \beta_1, \ldots, \beta_{|V_{B_2}| - 1}$ over $\mathbb{F}_q$ not all of them zero, such that,

$$v = \sum_{i=0}^{|V_{B_1}| - 1} \alpha_i u_i = \sum_{i=0}^{|V_{B_2}| - 1} \beta_i w_i,$$

or equivalently,

$$\sum_{i=0}^{|V_{B_1}| - 1} \alpha_i u_i - \sum_{i=0}^{|V_{B_2}| - 1} \beta_i w_i = 0.$$

Since $|B_1 \cup B_2| \leq k$, the columns of the set $V_{B_1} \cup V_{B_2}$ are linearly independent. Since $\sum_{i=0}^{|V_{B_1}| - 1} \alpha_i u_i - \sum_{i=0}^{|V_{B_2}| - 1} \beta_i w_i = 0$, we deduce that for all $u_i \in V_{B_1} \setminus V_{B_2}$, $\alpha_i = 0$ and for all $w_i \in V_{B_2} \setminus V_{B_1}$, $\beta_i = 0$. Therefore, $v \in \text{span} \left( V_{B_1} \cap V_{B_2} \right)$.

Next we will prove the following Lemma.

**Lemma 19.** For all $r$ such that $2 \leq r \leq |S_t|$, and $r$ distinct sets $B_1, B_2, \ldots, B_r \in S_t$,

$$( \bigcap_{1 \leq i \leq r} \text{span} V_{B_i} ) = \text{span} \left( \bigcap_{1 \leq i \leq r} V_{B_i} \right).$$

**Proof.** We prove this lemma by induction on the value of $r$. The base case was already proved in Claim \[\square\]

The step case for $r > 2$ is proved as follows. Suppose that

$$( \bigcap_{1 \leq i \leq r-1} \text{span} V_{B_i} ) = \text{span} \left( \bigcap_{1 \leq i \leq r-1} V_{B_i} \right),$$

and we will prove that

$$( \bigcap_{1 \leq i \leq r} \text{span} V_{B_i} ) = \text{span} \left( \bigcap_{1 \leq i \leq r} V_{B_i} \right).$$

Since

$$( \bigcap_{1 \leq i \leq r} \text{span} V_{B_i} ) = \left( \bigcap_{1 \leq i \leq r-1} \text{span} V_{B_i} \right) \cap \text{span} V_{B_r},$$

we use the induction assumption to get that

$$( \bigcap_{1 \leq i \leq r-1} \text{span} V_{B_i} ) \cap \text{span} V_{B_r} = \text{span} \left( \bigcap_{1 \leq i \leq r-1} V_{B_i} \right) \cap \text{span} V_{B_r},$$

and now we apply Claim \[\square\] on the sets $\bigcap_{1 \leq i \leq r-1} V_{B_i}$ and $V_{B_r}$ to get that

$$\text{span} \left( \bigcap_{1 \leq i \leq r-1} V_{B_i} \right) \cap \text{span} V_{B_r} = \text{span} \left( \bigcap_{1 \leq i \leq r} V_{B_i} \right),$$

which concludes the proof. \[\square\]
Next, for all \(1 \leq r \leq |S_t|\), let \(f(n,t,r,s)\) be the number of options to choose \(r\) sets from \(S_t\) where their intersection is of size \(s\), that is,

\[
f(n,t,r,s) = |\{\{B_1,\ldots,B_r\} \subseteq S_t \mid \bigcap_{1 \leq \ell \leq r} B_\ell = s\}|.
\]

The value of \(f(n,t,r,s)\) is calculated in the next lemma and its proof appears in Appendix E.

**Lemma 20.** For \(1 \leq r \leq |S_t|\), the value \(f(n,t,r,s)\) satisfies

\[
f(n,t,r,s) = \binom{n}{s} \sum_{m=0}^{t-s} (-1)^m \binom{n-s}{m} \binom{(n-s-m)}{r}.
\]

In the next claim, we list several combinatorial identities, which their proof is omitted as an exercise for the reader.

**Claim 9.**

(a) For all positive integer \(a\) it holds

\[
\sum_{i=1}^{a} (-1)^{i+1} \binom{a}{i} = 1.
\]

(b) For all positive integers \(a\) and \(b\) such that \(b \geq a\) it holds

\[
\sum_{i=0}^{a} (-1)^i \binom{b}{i} = (-1)^a \binom{b-1}{a}.
\]

(c) For all positive integers \(a, b\) and \(c\) such that \(c \geq b \geq a\) it holds

\[
\binom{c}{a} \binom{c-a}{b-a} = \binom{c}{b} \binom{b}{a}.
\]

(d) For all positive integers \(a, b\) and \(c\) such that \(c \geq b \geq a\) it holds

\[
\binom{c-a-1}{b-a} = \binom{c-a}{b-a} \frac{c-b}{c-a}.
\]

We are now ready to prove the main result of this section.

**Theorem 21.** For all positive integer \(n\), prime power \(q\) and even \(k\), \(t = k/2\), any optimal undirected \((n-k)\)-node-erasure-correcting code over \( \mathbb{F}_q \) satisfies

\[
q^{t^2} \geq \sum_{s=0}^{t} (-1)^{t-s} q^{s+1} \binom{n}{s} \binom{t}{s} \frac{n-t}{n-s}.
\]
Proof. Let \( C_U \) be an optimal undirected \((n-k)\)-node-erasure-correcting code over \( \mathbb{F}_q \) and let \( G \) be its generator matrix. We will use the inclusion-exclusion principle to calculate the number of columns that are not in \( \bigcup_{B \in S_t} \text{span} \, V_B \), where \( V_B \) is defined in (3.11).

We denote by \( W \) the number of options to choose every column of the matrix \( G \), that is, \( W = q^{k+1 \choose 2} = q^{2t+1 \choose 2} \). We will calculate \( |\bigcup_{B \in S_t} \text{span} \, V_B| \) by the inclusion-exclusion principle,

\[
| \bigcup_{B \in S_t} \text{span} \, V_B | = \sum_{r=1}^{\frac{|S_t|}{}} (-1)^{r+1} \left( \sum_{B_1, B_2, \ldots, B_r \in S_t} |\text{span} \, V_{B_1} \cap \cdots \cap \text{span} \, V_{B_r}| \right).
\]

Next, we present the following calculations and explain each step afterwards.

\[
| \bigcup_{B \in S_t} \text{span} \, V_B | = \\
= \sum_{r=1}^{\frac{|S_t|}{}} (-1)^{r+1} \left( \sum_{B_1, B_2, \ldots, B_r \in S_t} |\text{span} \, V_{B_1} \cap \cdots \cap \text{span} \, V_{B_r}| \right) \\
\overset{(I)}{=} \sum_{r=1}^{\frac{|S_t|}{}} (-1)^{r+1} \left( \sum_{B_1, B_2, \ldots, B_r \in S_t} |\text{span} \, (V_{B_1} \cap \cdots \cap V_{B_r})| \right) \\
\overset{(II)}{=} \sum_{r=1}^{\frac{|S_t|}{}} (-1)^{r+1} \left( \sum_{B_1, B_2, \ldots, B_r \in S_t} q^{s+1 \choose 2} \right) \\
\overset{(III)}{=} \sum_{r=1}^{\frac{|S_t|}{}} (-1)^{r+1} \left( \sum_{s=0 \atop \text{at least } s}^{t} q^{s+1 \choose 2} f(n, t, r, s) \right) \\
\overset{(IV)}{=} \sum_{r=1}^{\frac{|S_t|}{}} (-1)^{r+1} \left( \sum_{s=0}^{t} q^{s+1 \choose 2} \left( \sum_{m=0}^{n-s} (-1)^{m} \binom{n-s-m}{m} \right) \right) \left( \sum_{m=0}^{t-s} \binom{t-s-m}{r} \right)
\]
= \sum_{s=0}^{t} q^{(s+1)/2} \binom{n}{s} \left( \sum_{m=0}^{t-s} (-1)^m \binom{n-s}{m} \left( \sum_{r=1}^{\lfloor S_t \rfloor} (-1)^{r+1} \binom{n-s-m}{r} \right) \right) \\
(V) = \sum_{s=0}^{t} q^{(s+1)/2} \binom{n}{s} \left( \sum_{m=0}^{t-s} (-1)^m \binom{n-s}{m} \left( \sum_{r=1}^{\lfloor S_t \rfloor} (-1)^{r+1} \binom{n-s-m}{r} \right) \right) \\
(VI) = \sum_{s=0}^{t} q^{(s+1)/2} \binom{n}{s} \left( \sum_{m=0}^{t-s} (-1)^m \binom{n-s}{m} \right) \\
(VII) = \sum_{s=0}^{t} q^{(s+1)/2} \binom{n}{s} (-1)^t \binom{n-s-1}{t-s} \\
(VIII) = \sum_{s=0}^{t} (-1)^{t-s} q^{(s+1)/2} \binom{n}{t} \binom{n-s}{n-s} \\
(IX) = \sum_{s=0}^{t} (-1)^{t-s} q^{(s+1)/2} \binom{n}{t} \binom{n-t}{n-s}.

Equality (I) holds since by Lemma 19, \(|\bigcap_{1 \leq i \leq r} \text{span } V_{B_i}| = |\text{span } \bigcap_{1 \leq j \leq r} V_{B_i}|.

Equality (II) holds since for 0 \leq s \leq t, if |\bigcap_{1 \leq i \leq r} B_i| = s then |\text{span } \bigcap_{1 \leq i \leq r} B_i| = q^{(s+1)/2}. Equality (III) holds since by definition the number of options such that |\bigcap_{1 \leq i \leq r} B_i| = s is f(n, t, r, s). Equality (IV) holds by Lemma 20. Equality (V) holds since |S_t| = \binom{n}{t} > \binom{n-s-m}{t-s-m}, and for r > \binom{n-s-m}{t-s-m}, \binom{n-s-m}{r} will be zero. Equality (VI), (VII), (VIII), (IX) holds by Claim 9(a), 9(b), 9(c), 9(d), respectively.

Denote by E, the number of columns of \(F_q^{(s+1)/2}\) that are not in \(\bigcup_{B \in S} \text{span } V_B\), that is, \(E = W - |\bigcup_{B \in S} \text{span } V_B|\). Therefore, the code \(C_d\) exists only if \(E\) is not smaller than 0, or, \(W \geq |\bigcup_{B \in S} \text{span } V_B|\), that is,

\[ q^{(2t+1)/2} \geq \sum_{s=0}^{t} (-1)^{t-s} q^{(s+1)/2} \binom{n}{t} \binom{n-t}{n-s}. \]

\[ q^{(2t+1)/2} \geq \sum_{s=0}^{t} (-1)^{t-s} q^{(s+1)/2} \binom{n}{t} \binom{n-t}{n-s}. \]

For \(k = 2 (t = 1)\) the bound from Theorem 21 states that

\[ q^3 \geq \sum_{s=0}^{1} (-1)^{1-s} q^{(s+1)/2} \binom{n}{1} \binom{n-1}{n-s} \]

\[ = -n \cdot \frac{n-1}{n} + q \cdot n \cdot \frac{n-1}{n-1} \]

\[ = -n + 1 + q \cdot n = 1 + (q - 1)n, \]

57
or, \( q^3 - 1 \geq (q - 1)n \), that is equivalent to

\[
q^2 + q + 2 > n,
\]

which is the result of Theorem 18. For \( q = 2 \) we get that there is no binary code for \( n > 7 \). Similarly, it can be verified that for \( k = 4 \), there is no binary code for \( n > 20 \), and for \( k = 6 \) there is no binary code for \( n > 69 \).

In the next corollary, for fixed values of \( k \) and \( q \), we find an upper bound on the value of \( n \) for the existence of optimal undirected \((n - k)\)-node-erasure-correcting code over \( \mathbb{F}_q \).

**Corollary 1.** Let \( k \) be a fixed positive integer and let \( q \) be a prime power. For all \( n \) such that \( \log_q(n) > \frac{3}{2}k + \log_q(k) + \frac{3}{2} \), there does not exist an optimal undirected \((n - k)\)-node-erasure-correcting code over \( \mathbb{F}_q \).

**Proof.** We present the following calculations and explain each step afterwards. From Theorem 21 we get that for \( t = k/2 \) the following inequality holds.

\[
q^{\left(\frac{2t+1}{2}\right)} \geq \sum_{s=0}^{t} (-1)^{t-s} q^{\left(\frac{s+1}{2}\right)} \binom{n}{t-s} \frac{n-t}{n-s}
\]

\[
= q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - \binom{n}{t} \sum_{s=0}^{t-1} (-1)^{t-s} q^{\left(\frac{s+1}{2}\right)} \binom{t}{s} \frac{n-t}{n-s}
\]

\[
\geq q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - \binom{n}{t} \sum_{s=0}^{t-1} (-1)^{t-s} q^{\left(\frac{s+1}{2}\right)} \binom{t}{s}
\]

\[
\geq q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - q^{\left(\frac{t}{2}\right)} \binom{n}{t} \sum_{s=0}^{t-1} \left( \frac{t}{t-s} \right) \text{ if } (t-s) \text{ is even}
\]

\[
\geq q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - q^{\left(\frac{t}{2}\right)} \binom{n}{t} \sum_{i=0}^{[t/2]} \left( \frac{t}{2i} \right)
\]

\[
= q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - q^{\left(\frac{t}{2}\right)} \binom{n}{t} 2^{t-1}
\]

\[
\geq q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - \frac{1}{2} q^{\left(\frac{t}{2}\right)} \binom{n}{t} q^t
\]

\[
= q^{\left(\frac{t+1}{2}\right)} \binom{n}{t} - \frac{1}{2} q^{\left(\frac{t+1}{2}\right)} \binom{n}{t}
\]

\[
= \frac{1}{2} q^{\left(\frac{t+1}{2}\right)} \binom{n}{t}.
\]
Inequality (I) holds since $0 \leq s \leq t$, so $\frac{n-t}{n-s} \leq 1$. Inequality (II) holds since we remove all the cases where $(-1)^{t-s}$ is negative. Inequality (III) holds since for all $s \in [t]$, $q^{s+t} \leq q^t$ and $(\frac{t}{s}) = (\frac{t}{t-s})$. Equality (IV) holds based upon the formula $\sum_{i=0}^{\frac{t}{2}} (\frac{t}{2i}) = 2^{t-1}$. Inequality (V) holds since $2^t \leq q^t$.

Therefore, we deduce that for any optimal undirected $(n-k)$-node-erasure-correcting code over $\mathbb{F}_q$,

$$q^{\frac{k+1}{2}} \geq \frac{1}{2} q^{\left(\frac{k+1}{2}\right)} \left(\frac{n}{k/2}\right).$$

Since for positive integer $a$ and fixed $b$, such that $b \leq a$, $\binom{a}{b} \geq \frac{(a-b)b}{b!}$, we get that

$$q^{\frac{k+1}{2}} - \left(\frac{k+1}{2}\right) \geq \frac{1}{2} \left(\frac{n}{k/2}\right) \geq \frac{(n-k)^{k/2}}{2^{k/2}!},$$

$$q^{\frac{3}{2}k^2 + \frac{3}{2}k} 2(k/2)! \geq (n-k)^{k/2},$$

$$q^{\frac{3}{2}k^{1/2} \frac{k/2}{2}(k/2)!} \geq n - k,$$

$$q^{\frac{3}{2}k^{1/2} \frac{k/2}{2}(k/2)! + k} \geq n.$$

Since $\frac{k}{2}(k/2)! \leq 2^{k/2}(k/2)! \leq 2k/2$ we can write

$$q^{\frac{3}{2}k^{1/2} + k} + k \geq n,$$

$$(q^{\frac{3}{2}k^{1/2} + 1})k \geq n.$$

and

$$\log_q(q^{\frac{3}{2}k^{1/2} + 1} + \log_q(k)) \geq \log_q(n).$$

Since for all $a > 1$, $\log_q(a) + 1 > \log_q(a + 1)$, we conclude that

$$\frac{3}{4}k + \frac{3}{2} + \log_q(k) \geq \log_q(n).$$

Therefore, for all $n$ such that $\log_q(n) > \frac{3}{4}k + \log_q(k) + \frac{3}{2}$ an optimal undirected $(n-k)$-node-erasure-correcting code over $\mathbb{F}_q$ does not exist.

For odd $k$ by the same method we can deduce that for all $n$ satisfying the same inequality $\log_q(n) > \frac{3}{4}k + \log_q(k) + \frac{3}{2}$ an optimal undirected $(n-k)$-node-erasure-correcting code over $\mathbb{F}_q$ does not exist. Moreover, for directed graphs and any fixed $k$, for all $n$ satisfying the inequality $\log_q(n) > \frac{3}{2}k + \log_q(k) + 1$ an optimal directed $(n-k)$-node-erasure-correcting code over $\mathbb{F}_q$ does not exist.
3.8 Conclusion

In this paper we proposed a new construction of codes, called codes over graphs. We studied here complete undirected or directed graphs and the goal was to construct codes over graphs which are capable to correct the erasure of node failures. We built upon previous constructions of product codes and rank metric codes. The former set of codes provided us with optimal codes with linear field size and the latter was used for the construction of non-optimal binary codes. We were then aspired by the construction of EVENODD codes in order to construct optimal codes over graphs correcting two node failures over the binary field. Lastly, we studied upper bounds on the number of nodes of optimal codes.

Acknowledgments

The authors would like to thank Jehoshua Bruck for valuable discussions and Ron M. Roth for his contribution to the results in Section 3.7 and sharing reference [16] with them.

Appendix A

Claim [1] For all $h \in [n - 1]$, $|S_h| = n - 1$ and for all $m \in [n]$, $|D_m| = \frac{n+1}{2}$.

Proof. The first part of the claim is readily verified. For $m \in [n]$, by the definition of $D_m$,

$$\{ \langle v_k, v_\ell \rangle \mid k, \ell \in [n] \setminus \{n - 2\}, (k + \ell)_n = m \} \cap \{\langle v_{n-1}, v_{n-2} \rangle \} = \emptyset.$$ 

Note that,

$$|\{\langle v_k, v_\ell \rangle \mid k, \ell \in [n] \setminus \{n - 2\}, (k + \ell)_n = m \}|$$

$$= |\{\langle v_k, v_{(m-k)_n} \rangle \mid k, (m-k)_n \in [n] \setminus \{n - 2\}\}|$$

$$= |\{\langle v_k, v_{(m-k)_n} \rangle \mid k \in [n] \setminus \{n - 2, (m+2)_n\}\}|.$$ 

If $(m+2)_n \neq n-2$ then $k$ gets $n-2$ distinct values and there are $n-2$ options for edge $(v_k, v_{(m-k)_n})$. For each of the $n-2$ options either $k \neq (m-k)_n$ and we get each edge counted twice since $(v_k, v_{(m-k)_n}) = (v_{(m-k)_n}, v_k)$, or $k = (m-k)_n$ and we get the self loop $(v_k, v_k)$. Therefore,

$$|\{\langle v_k, v_{(m-k)_n} \rangle \mid k \in [n] \setminus \{n - 2, (m+2)_n\}\}| = \frac{n-3}{2} + 1 = \frac{n-1}{2}.$$ 

If $(m+2)_n = n-2$ then $k$ gets $n-1$ distinct values and there are $n-1$ options for edge $(v_k, v_{(m-k)_n})$. For each of the $n-1$ options if $k \neq (m-k)_n$ we get each edge
counted twice since \( \langle v_k, v_{(m-k)n} \rangle = \langle v_{(m-k)n}, v_k \rangle \). Notice that there is no option for self loop \( \langle v_k, v_k \rangle \) since it can be generated only for \( k = n - 2 \). Therefore, in this case we also have that

\[
|\{ \langle v_k, v_{(m-k)n} \rangle \mid k \in [n] \setminus \{n-2\} \}| = \frac{n-1}{2}.
\]

Therefore for all \( m \in [n] \) we have that

\[
|D_m| = |\{ \langle v_k, v_\ell \rangle \mid k, \ell \in [n] \setminus \{n-2\}, \langle k + \ell \rangle_n = m \}| +
|\{ \langle v_{n-1}, v_{n-2} \rangle \}| = \frac{n-1}{2} + 1 = \frac{n+1}{2}.
\]

\( \square \)

### Appendix B

**Claim** [2] The sets \( S_h, D_m, F_t \) satisfy the following properties.

(a) For all distinct \( h, i \in [n-2] \), \( S_h \cap F_i = \{\langle v_h, v_i \rangle\} \).

(b) For all \( i \in [n-2] \), \( S_{i-2} \cap F_i = \{\langle v_i, v_i \rangle\} \).

(c) For all pairwise distinct \( i, j, h \in [n-2] \), \( S_h \cap (F_i \cup F_j) = \{\langle v_h, v_i \rangle, \langle v_h, v_j \rangle\} \).

(d) For all distinct \( i, j \in [n-2] \), \( S_{i-2} \cap (F_i \cup F_j) = \{\langle v_i, v_i \rangle, \langle v_j, v_j \rangle\} \).

(e) For all \( i \in [n-2] \), \( D_{i-2} \cap F_i = \emptyset \) and for \( i = n-1 \), \( D_{n-3} \cap F_{n-1} = \{\langle v_{n-2}, v_{n-1} \rangle\} \).

(f) For all \( i \in [n-2] \), \( s \in [n] \setminus \{i-2\}_n \), \( D_s \cap F_i = \{\langle v_{(s-i)_n}, v_i \rangle\} \). For \( i \) such that \( i = n-1 \), \( D_s \cap F_{n-1} = \{\langle v_{(s+1)_n}, v_{n-1} \rangle, \langle v_{n-2}, v_{n-1} \rangle\} \).

(g) For all distinct \( i, j \in [n-2] \), \( D_{i+j} \cap F_j = \{\langle v_i, v_j \rangle\} \).

(h) For all distinct \( i, j \in [n-2] \), \( D_{i+j} \cap (F_i \cup F_j) = \{\langle v_{(i+j-2)_n}, v_i \rangle\} \).

(i) For all distinct \( i, j \in [n-2] \), \( D_{i+j} \cap (F_i \cup F_j) = \{\langle v_i, v_j \rangle\} \).

(j) For all \( i \in [n-2] \), \( D_{i-2} \cap (F_i \cup F_{n-2}) = \{\langle v_{n-1}, v_{n-2} \rangle\} \).

**Proof.** (a) Assume that there is an edge \( \langle v_h, v_\ell \rangle \in S_h \cap F_i \). Since \( \langle v_h, v_\ell \rangle \in F_i \) such that \( h \neq i \), deduce that \( \ell = i \). Therefore \( S_h \cap F_i = \{\langle v_h, v_i \rangle\} \).

(b) Assume that there is an edge \( \langle v_k, v_k \rangle \in S_{i-2} \cap F_i \), thus, \( k = i \). Therefore \( S_{i-2} \cap F_i = \{\langle v_i, v_i \rangle\} \).

(c) Using (a) deduce,

\[
S_h \cap (F_i \cup F_j) = (S_h \cap F_i) \cup (S_h \cap F_j) = \{\langle v_h, v_i \rangle, \langle v_h, v_j \rangle\}.
\]
(d) Using \(\text{[4]}\), deduce,

\[
S_{n-2} \cap (F_i \cup F_j) = \\
(S_{n-2} \cap F_i) \cup (S_{n-2} \cap F_j) = \{\langle v_i, v_i \rangle, \langle v_j, v_j \rangle\}
\]

(e) For \(i < n - 2\), \(\langle v_{n-1}, v_{n-2} \rangle \notin F_i\), it is enough to show that

\[
\{\langle v_k, v_\ell \rangle \mid k, \ell \in [n] \setminus \{n - 2\}, (k + \ell)_n = \langle i - 2 \rangle_n \}
\]

\[
\{\langle v_i, v_\ell \rangle \mid \ell \in [n] \}
\]

Assume that there is an edge \(\langle v_k, v_\ell \rangle \in D_{(i-2)_n} \cap F_i\), where \(k, \ell \in [n] \setminus \{n - 2\}\) and in particular \(k, \ell \neq n - 2\). According to the definition of the set \(F_i\) we have that \(k = i\) or \(\ell = i\). Without loss of generality assume that \(k = i\) and according to the definition of the set \(D_{(i-2)_n}\) we have that \((i + \ell)_n = (i - 2)_n\) and thus \(\ell = n - 2\), in contradiction. By the same concept, it is easy to verify that for \(i = n - 1\), the only edge in intersection of \(D_{n-3}\) and \(F_{n-1}\) is \(\langle v_{n-1}, v_{n-2} \rangle\).

(f) Since \(i < n - 2\), \(\langle v_{n-1}, v_{n-2} \rangle \notin F_i\), so it is enough to show that

\[
\{\langle v_k, v_\ell \rangle \mid k, \ell \in [n] \setminus \{n - 2\}, (k + \ell)_n = s \}
\]

\[
\{\langle v_i, v_\ell \rangle \mid \ell \in [n] \}
\]

Assume that there is an edge \(\langle v_k, v_\ell \rangle \in D_s \cap F_i\). According to the definition of the set \(F_i\) we have that \(k = i\) or \(\ell = i\). Without loss of generality assume that \(k = i\). By definition of the set \(D_s\) we deduce that \((i + \ell)_n = s\), so \(\ell = (s - i)_n\). In case where \(i = n - 1\), we simply add the \(\langle v_{n-1}, v_{n-2} \rangle\) edge to the intersection between \(D_s\) and \(F_{n-1}\), and the remaining proof is similar.

(g) Let \(s = \langle i + j \rangle_n\). Since \(j < n - 2\), \(s = \langle i + j \rangle_n \neq \langle i - 2 \rangle_n\), therefore according to \(\text{[4]}\) we deduce that \(D_{(i+j)_n} \cap F_i = \{\langle v_i, v_j \rangle\}\).

(h) Let \(s = \langle j - 2 \rangle_n\). Since \(i \neq j\), \(s = \langle j - 2 \rangle_n \neq \langle i - 2 \rangle_n\), therefore according to \(\text{[4]}\) we deduce that \(D_{(j-2)_n} \cap F_i = \{\langle v_{(j-i-2)_n}, v_i \rangle\}\), and since \(j \in [n - 2]\), by \(\text{[4]}\) deduce that \(D_{(j-2)_n} \cap F_j = \emptyset\). Therefore:

\[
D_{(j-2)_n} \cap (F_i \cup F_j) = \\
(D_{(j-2)_n} \cap F_i) \cup (D_{(j-2)_n} \cap F_j) = \{\langle v_{(j-i-2)_n}, v_i \rangle\}
\]

(i) Let \(s = \langle i + j \rangle_n\), therefore,

\[
D_{(i+j)_n} \cap (F_i \cup F_j) = (D_{(i+j)_n} \cap F_i) \cup (D_{(i+j)_n} \cap F_j) = \\
\{\langle v_i, v_j \rangle\} \cup \{\langle v_j, v_i \rangle\} = \{\langle v_i, v_j \rangle\}
\]

where equality holds according to \(\text{[4]}\).
(j) Since \( i \in [n - 2] \), by (e) we get that \( D_{(i-2)} \cap F_i = \emptyset \). Since the set \( D_{(i-2)} \) has only one edge connecting the node \( v_{n-2} \), we get that \( D_{(i-2)} \cap F_{n-2} = \{ \langle v_{n-1}, v_{n-2} \rangle \} \), and therefore:

\[
D_{(i-2)} \cap (F_i \cup F_{n-2}) = (D_{(i-2)} \cap F_i) \cup (D_{(i-2)} \cap F_{n-2}) = \{ \langle v_{n-1}, v_{n-2} \rangle \}.
\]

\[\Box\]

### Appendix C

#### Claim 3

The following properties hold:

(a) \( x \neq y \) and \( x + y = n - 2 \).

(b) \( s_1^{(x)} = \tilde{s}_1^{(y)} = n - 1 \).

(c) \( i, j \in A \) or \( i, j \in B \) but not in both.

(d) \( n - 2 \notin A \cup B \).

(e) \( A \cap B = \{n - 1\} \).

(f) \( |A| = x + 1 \) and \( |B| = y + 1 \).

(g) \( s_1^{(0)} \neq i \) and \( \tilde{s}_1^{(0)} \neq j \).

(h) For \( 0 < t \leq x \), \( s_1^{(t-1)} = j \) if and only if \( s_1^{(t)} = i \).

(i) For \( 0 < t \leq x \), \( s_2^{(t)} \notin \{\langle i - 2 \rangle_n, \langle j - 2 \rangle_n\} \).

**Proof.** First we remind that \( n \) is a prime number.

(a) Since \( x = \langle -1 - d^{-1} \rangle_n \) and \( y = \langle -1 + d^{-1} \rangle_n \), we get

\[
x - y = \langle -1 - d^{-1} \rangle_n - \langle -1 + d^{-1} \rangle_n = \langle -2d^{-1} \rangle_n.
\]

Since \( n \) is a prime number, we deduce that \( \langle -2d^{-1} \rangle_n \neq 0 \) and so \( x \neq y \). By definition of \( x \) and \( y \),

\[
\langle x + y \rangle_n = \langle -1 - d^{-1} \rangle_n + \langle -1 + d^{-1} \rangle_n = \langle -2 \rangle_n.
\]

Since \( x, y \in [n] \) and \( x \neq y \) we conclude that \( x + y = n - 2 \).

(b) According to the definition of \( s_1^{(t)} \) we get for \( t = x \) that

\[
s_1^{(x)} = \langle -d(x + 1) - 2 \rangle_n = \langle -d(-d^{-1}) - 2 \rangle_n = \langle 1 - 2 \rangle_n = n - 1.
\]

The proof that \( \tilde{s}_1^{(y)} = n - 1 \) is identical.
(c) Assume that \( j \in A \). Since \( j \neq n - 1 = s_1^{(x)} \), there exists \( t_1 < x \) such that 
\[
s_1^{(t_1)} = \langle -d(t_1 + 1) - 2 \rangle_n = j.
\]
Hence,
\[
s_1^{(t_1+1)} = \langle s_1^{(t_1)} - d \rangle_n = \langle j - (j - i) \rangle_n = i
\]
and we get that \( i \in A \). The proof that if \( i \in A \) then \( j \in A \) is similar.
Assume again that \( j \in A \), and on contrary that \( i \in B \), Since \( i \neq n - 1 = s_1^{(y)} \),
there exists \( t_2 < y \) such that \( s_1^{(t_2)} = i \), that is
\[
\langle d(t_2 + 1) - 2 \rangle_n = i.
\]
Therefore
\[
d = j - i = \langle -d(t_1 + 1) - 2 \rangle_n - \langle d(t_2 + 1) - 2 \rangle_n,
\]
and
\[
\langle d(t_1 + t_2 + 3) \rangle_n = 0
\]
which leads to a contradiction since \( t_1 + t_2 + 3 \leq x - 1 + y - 1 + 3 = n - 1 \).
Similarly we prove that if \( i \notin B \) then \( j \in A \).

(d) We will prove without loss of generality that \( n - 2 \notin A \). Assume in contrary that \( n - 2 \in A \), then there exists \( 0 \leq t \leq x \) such that 
\[
s_1^{(t)} = \langle -d(t + 1) - 2 \rangle_n = n - 2.
\]
Therefore, \( \langle d(t + 1) \rangle_n = 0 \), which leads to a contradiction.

(e) By [4], \( n - 1 \in A \cap B \). Assume on contrary that exists \( h \neq n - 1 \) such that \( h \in A \cap B \). Since \( h \neq n - 1 = s_1^{(x)} = \tilde{s}_1^{(y)} \), there exist \( t_1 < x, t_2 < y \) such that
\[
h = \langle -d(t_1 + 1) - 2 \rangle_n = \langle d(t_2 + 1) - 2 \rangle_n.
\]
Hence we get
\[
\langle d(t_1 + t_2 + 2) \rangle_n = 0,
\]
and again we get a contradiction.

(f) Assume that \( |A| < x + 1 \). Therefore there are \( 0 \leq t_2 < t_1 \leq x \) such that 
\[
s_1^{(t_1)} = s_1^{(t_2)}.
\]
By definition of \( s_1^{(t_1)}, s_1^{(t_2)} \) we deduce,
\[
\langle -d(t_1 + 1) - 2 \rangle_n = \langle -d(t_2 + 1) - 2 \rangle_n.
\]
Hence we get
\[
\langle d(t_1 - t_2) \rangle_n = 0,
\]
and since \( 0 < t_1 - t_2 \leq x \leq n - 2 \) we get a contradiction. The \( |B| = y + 1 \) is proved similarly.
(g) Assume that \( s_1^{(0)} = i \), therefore
\[
\langle -d - 2 \rangle_n = \langle i - j - 2 \rangle_n = i,
\]
\[
\langle n - j - 2 \rangle_n = 0,
\]
\[
n - 2 = j,
\]
and that is a contradiction. The \( \tilde{s}_1^{(0)} \neq j \) is proved similarly.

(h) If \( s_1^{(t-1)} = j \) then,
\[
s_1^{(t)} = s_1^{(t-1)} - d = j - (j - i) = i.
\]
If \( s_1^{(t)} = i \) then,
\[
s_1^{(t-1)} = s_1^{(t)} + d = i + (j - i) = j.
\]

(i) If \( s_2^{(t)} = \langle j - 2 \rangle_n \) then
\[
s_1^{(t)} = \langle s_2^{(t)} - j \rangle_n = \langle (j - 2) - j \rangle_n = n - 2,
\]
and we know that \( s_1^{(t)} \neq n - 2 \) by \((f)\). If \( 0 < t \leq x \) and \( s_2^{(t)} = \langle i - 2 \rangle_n \) then
\[
s_1^{(t)} = \langle s_2^{(t)} - j \rangle_n = \langle (i - 2) - j \rangle_n = \langle -d - 2 \rangle_n = s_1^{(0)},
\]
but since \( |A| = x + 1 \), \( s_1^{(t)} = s_1^{(0)} \) only for \( t = 0 \) and that is a contradiction.

\( \Box \)

Appendix D

Claim 4. The following properties hold:

(a) \( V_1 \cap V_2 = \emptyset \).

(b) \( |V| = 2n - 4 \).

(c) \( \langle v_i, v_j \rangle \notin V \).

(d) \( \langle v_{n-2}, v_i \rangle \notin V \) and \( \langle v_{n-2}, v_j \rangle \notin V \).

Proof. (a) According to Claim 3, \( A \cap B = \{ n - 1 \} \) and we get
\[
\{ \langle v_{s_1}, v_i \rangle, \langle v_{s_1}, v_j \rangle : s_1 \in A \setminus \{ n - 1 \} \} \cap
\{ \langle v_{\tilde{s}_1}, v_i \rangle, \langle v_{\tilde{s}_1}, v_j \rangle : \tilde{s}_1 \in B \setminus \{ n - 1 \} \} = \emptyset.
\]
By the definition of \( V_1 \) and \( V_2 \),
\[
\{ \langle v_{n-1}, v_j \rangle, \langle v_i, v_i \rangle, \langle v_j, v_j \rangle \} \subseteq V_1 \setminus V_2,
\]
and \( \langle v_{n-1}, v_i \rangle \in V_2 \setminus V_1 \), therefore \( V_1 \cap V_2 = \emptyset \).
(b) Since \(|A| = x + 1\) and since \(i, j \in A\) we get that,

\[
\{|\{v_{s_1}, v_i\}, \langle v_{s_1}, v_j \rangle : s_1 \in A \setminus \{n - 1\}\}|
\]

\[
= 2(|A| - 1) - 2 = 2(x + 1 - 1) - 2 = 2x - 2,
\]

thus, \(|V_1| = 2x - 1\). Similarly, since \(|B| = y + 1\) and since \(i, j \notin B\) we get that,

\[
\{|\{v_{\bar{s}_1}, v_j\}, \langle v_{\bar{s}_1}, v_i \rangle : \bar{s}_1 \in B \setminus \{n - 1\}\}|
\]

\[
= 2(|B| - 1) = 2(y + 1 - 1) = 2y,
\]

thus, \(|V_2| = 2y + 1\). By (a), \(V_1 \cap V_2 = \emptyset\) we deduce,

\[
|V| = |V_1| + |V_2| = (2x - 1) + (2y + 1) = 2(n - 2) = 2n - 4.
\]

(c) According to the definition of \(V_1\) and \(V_2\), \(\langle v_i, v_j \rangle \notin V_1\) and \(\langle v_i, v_j \rangle \notin V_2\).

(d) By Claim 3(d), \(n - 2 \notin A \cup B\), therefore by the definition of \(V_1\) and \(V_2\), \(\langle v_{n-2}, v_j \rangle, \langle v_{n-2}, v_i \rangle \notin V_1\) and \(\langle v_{n-2}, v_j \rangle, \langle v_{n-2}, v_i \rangle \notin V_2\). \(\square\)

Appendix E

**Lemma 20** For \(1 \leq r \leq |S_t|\), the value \(f(n, t, r, s)\) satisfies

\[
f(n, t, r, s) = \binom{n}{s} \sum_{m=0}^{t-s} (-1)^m \binom{n-s}{m} \binom{n-s-m}{r-m}.
\]

**Proof.** Note that \(f(n, t, r, s)\) is the cardinality of the set

\[
H = \{\{B_1, \ldots, B_r\} \subseteq S_t \mid | \bigcap_{1 \leq \ell \leq r} B_\ell | = s\}.
\]

For a fixed set \(B \subseteq [n]\) (e.g., \(B = [s]\)), we denote by \(H_B\) the set

\[
H_B = \{\{B_1, \ldots, B_r\} \subseteq S_t \mid | \bigcap_{1 \leq \ell \leq r} B_\ell | = B\}.
\]

Since there are \(\binom{n}{s}\) options to choose the set \(B\), we deduce that

\[
|H| = \sum_{B \subseteq [n]} |H_B| = \binom{n}{s} |H_{[s]}|.
\]

For the remainder of the proof, we find the side of the set \(H_{[s]}\).
We denote the set
\[ W = \{ \{B_1, \ldots, B_r\} \subseteq S_t \mid [s] \subseteq \bigcap_{1 \leq \ell \leq r} B_\ell \}, \]
where \(|W| = \binom{n-s}{t-s-r} \). For \( i \in ([n] \setminus [s]) \) we also define \( A_i \) as follows
\[ A_i = \{ \{B_1, \ldots, B_r\} \in W \mid i \in \bigcap_{1 \leq \ell \leq r} B_\ell \}, \]
where it holds that \(|H| = |W| - |\bigcup_{i \in ([n] \setminus [s])} A_i| \). Therefore, we calculate \(|\bigcup_{i \in ([n] \setminus [s])} A_i|\) by using the inclusion-exclusion principle
\[ |\bigcup_{i \in ([n] \setminus [s])} A_i| = \sum_{m=1}^{n-s} (-1)^{m+1} \left( \sum_{i_1 < \cdots < i_m \leq n-1} |A_{i_1} \cap \cdots \cap A_{i_m}| \right). \]
For all \( 1 \leq m \leq n-s \) and \( 0 \leq i_1 < \cdots < i_m \leq n-1 \), it holds
\[ |A_{i_1} \cap \cdots \cap A_{i_m}| = \binom{n-s-m}{r}, \]
since the intersection \( A_{i_1} \cap \cdots \cap A_{i_m} \) includes at least the set \([s] \cup \{i_1, \ldots, i_m\}\). Notice that for \( m > t-s \), \(|A_{i_1} \cap \cdots \cap A_{i_m}| = 0\) so we can take only the cases where \( 1 \leq m \leq t-s \). Therefore we can write
\[ |\bigcup_{i \in ([n] \setminus [s])} A_i| = \sum_{m=1}^{t-s} (-1)^{m+1} \left( \sum_{i_1 < \cdots < i_m \leq n-1} |A_{i_1} \cap \cdots \cap A_{i_m}| \right) = \sum_{m=1}^{t-s} (-1)^{m+1} \left( \binom{n-s-m}{m} \right) \binom{n-s-m}{r}. \]
Finally, we conclude that
\[ |H| = |W| - |\bigcup_{i \in ([n] \setminus [s])} A_i| = \sum_{m=0}^{t-s} (-1)^{m} \left( \binom{n-s-m}{m} \right) \binom{n-s-m}{r}, \]
and since \( f(n, t, r, s) = |H| \) we get
\[ f(n, t, r, s) = \binom{n}{s} |H| = \binom{n}{s} \sum_{m=0}^{t-s} (-1)^{m} \left( \binom{n-s-m}{m} \right) \binom{n-s-m}{r}. \]
Bibliography


Chapter 4

Double and Triple Node-Erasure-Correcting Codes over Graphs

Lev Yohananov, Yuval Efron, and Eitan Yaakobi

Abstract

In this paper we study array-based codes over graphs for correcting multiple node failures. These codes have applications to neural networks, associative memories, and distributed storage systems. We assume that the information is stored on the edges of a complete undirected graph and a node failure is the event where all the edges in the neighborhood of a given node have been erased. A code over graphs is called ρ-node-erasure-correcting if it allows to reconstruct the erased edges upon the failure of any $\rho$ nodes or less. We present a binary optimal construction for double-node-erasure correction together with an efficient decoding algorithm, when the number of nodes is a prime number. Furthermore, we extend this construction for triple-node-erasure-correcting codes when the number of nodes is a prime number and two is a primitive element in $\mathbb{Z}_n$. These codes are at most a single bit away from optimality.

4.1 Introduction

Networks and distributed storage systems are usually represented as graphs with the information stored in the nodes (vertices) of the graph. In our recent work [23, 24, 25], we have introduced a new model which assumes that the information is stored on the edges. This setup is motivated by several information systems. For example, in neural networks, the neural units are connected via links which store and transmit information between the neural units [10]. Simi-
larly, in associative memories, the information is stored by associations between different data items [21]. Furthermore, representing information in a graph can model a distributed storage system [7] while every two nodes can be connected by a link that represents the information that is shared by the nodes.

In [23, 24, 25], we introduced the notion of codes over graphs, which is a class of codes storing the information on the edges of a complete undirected graph (including self-loops). Thus, each codeword is a labeled graph with \( n \) nodes (vertices) and each of the \( \binom{n+1}{2} \) edges stores a symbol over an alphabet \( \Sigma \). A node failure is the event where all the edges incident with a given node have been erased, and a code over graphs is called \( \rho \)-node-erasure-correcting if it allows to reconstruct the contents of the erased edges upon the failure of any \( \rho \) nodes or less.

The information stored in a complete undirected graph can be represented by an \( n \times n \) symmetric array and a failure of the \( i \)th node corresponds to the erasure of the \( i \)th row and \( i \)th column in the array. Hence, this problem is translated to the problem of correcting symmetric crisscross erasures in square symmetric arrays [15]. By the Singleton bound, the number of redundancy edges (i.e., redundancy symbols in the array) of every \( \rho \)-node-erasure-correcting code must be at least \( n\rho - \binom{\rho}{2} \), and a code meeting this bound will be referred as optimal. While the construction of optimal codes is easily accomplished by MDS codes, their alphabet size must be at least the order of \( n^2 \), and the task of constructing optimal (or close to optimal) codes over graphs over smaller alphabets remains an intriguing problem.

A natural approach to address this problem is by using the wide existing knowledge on array code constructions such as [2, 4, 5, 9, 12, 13, 16, 17, 18, 20]. However, the setup of codes over graphs differs from that of classical array codes in two respects. First, the arrays are symmetric, and, secondly, a failure of the \( i \)th node in the graph corresponds to the failure of the \( i \)th row and the \( i \)th column (for the same \( i \)) in the array. Most existing constructions of array codes are not designed for symmetric arrays, and they do not support this special row–column failure model. However, it is still possible to use existing code constructions and modify them to the special structure of the above erasure model in graphs, as was done in [23, 24, 25]. More specifically, based upon product codes [11, 8], a construction of optimal codes whose alphabet size grows only linearly with \( n \) has been proposed. Additionally, using rank-metric codes [13, 16, 14], binary codes over graphs were designed, however they are relatively close—yet do not attain—the Singleton bound. In [23, 24], a construction of optimal binary codes for two node failures was also presented based upon ideas from EVENODD codes [2].

In this paper we build upon some of the methods that were used in the array code constructions. An example of such an approach is the algebraic representation of EVENODD codes [2]. Another similar construction for optimal \( (p-1) \times p \) array codes that can tolerate any \( \rho \) column erasures was given in [5] as well as its extensions in [4] and [11]. In these papers, the authors also used an algebraic
approach, where each column of the array code is represented as a symbol over a fixed ring, which is then interpreted as a linear MDS code of length \( p \) over the ring. Note that these constructions cannot be used directly for the problem studied in the paper for the reasons mentioned above. However, we still show how to take advantage of this algebraic approach in order to construct double- and triple-node-erasure-correcting codes.

Another approach for handling symmetric crisscross erasures (in symmetric arrays) is by using symmetric rank-metric codes. In [16], Schmidt presented a construction of linear \([n \times n, k, d]\) symmetric binary array codes with minimum rank \( d \), where \( k = n(n-d+2)/2 \) if \( n-d \) is even, and \( k = (n+1)(n-d+1)/2 \) otherwise. Such codes can correct any \( d-1 \) column or row erasures. Hence, it is possible to use these codes to derive \( \rho \)-node-failure-correcting codes while setting \( d = 2\rho+1 \), as the \( \rho \) node failures translate into the erasure of \( \rho \) columns and \( \rho \) rows. However, the redundancy of these codes is \( \binom{\rho}{2} \) symbols away from the Singleton bound for symmetric crisscross erasures (e.g., for \( \rho = 2 \), their redundancy is \( 2n \) while the Singleton lower bound is \( 2n - 1 \)).

In this paper we carry an algebraic approach such as the one presented in [4, 5], and [11] in order to propose new constructions of binary codes over graphs. In Section 4.2, we formally define codes over graphs and review several basic properties from [23, 25] that will be used in the paper. In Section 4.3, we present our optimal binary construction for two-node failures along with its decoding procedure. This construction is not only simpler than the one given in [23, 25], but it also provides a good intuition to understand the triple-node-erasure-correcting codes in the paper. Furthermore, in Section 4.4, it is shown how to efficiently decode the case of a single node failure for this construction. Then, in Section 4.5, we extend this construction for the three-node failures case. This new construction is only at most a single bit away from the Singleton bound, thereby outperforming the construction obtained from [16]. In Section 4.6, we show how to efficiently decode the failure of three nodes. Lastly, Section 4.7 concludes the paper.

### 4.2 Definitions and Preliminaries

For a positive integer \( n \), the set \( \{0, 1, \ldots, n-1\} \) will be denoted by \([n]\) and for a prime power \( q \), \( \mathbb{F}_q \) is the finite field of size \( q \). A linear code of length \( n \) and dimension \( k \) over \( \mathbb{F}_q \) will be denoted by \([n, k]_q\) or \([n, k, d]_q\), where \( d \) denotes its minimum distance. In the rest of this section, we follow the definitions of our previous work [23] for codes over graphs.

A graph will be denoted by \( G = (V_n, E) \), where \( V_n = \{v_0, v_1, \ldots, v_{n-1}\} \) is its set of \( n \) nodes (vertices) and \( E \subseteq V_n \times V_n \) is its edge set. In this paper, we only study complete undirected graphs with self-loops, and in this case, the edge set of an undirected graph \( G \) over an alphabet \( \Sigma \) is defined by \( E = \{(v_i, v_j) \mid (v_i, v_j) \in V_n \times V_n, i \geq j\} \), with a labeling function \( L : V_n \times V_n \rightarrow \Sigma \). By a slight abuse
of notation, every undirected edge in the graph will be denoted by \( \langle v_i, v_j \rangle \) where the order in this pair does not matter, that is, the notation \( \langle v_j, v_i \rangle \) is identical to the notation \( \langle v_i, v_j \rangle \), and thus there are \( \binom{n+1}{2} \) edges. We will use the notation \( G = (V_n, L) \) for such graphs. For the rest of the paper, whenever we refer to a graph we refer to an undirected graph.

The labeling matrix of an undirected graph \( G = (V_n, L) \) is an \( n \times n \) symmetric matrix over \( \Sigma \) denoted by \( A_G = [a_{i,j}]_{i=0,j=0}^{n-1,n-1} \), where \( a_{i,j} = L(v_i, v_j) \). We also use the lower-triangle-labeling matrix of \( G \) to be the \( n \times n \) matrix \( A'_G = [a'_{i,j}]_{i=0,j=0}^{n-1,n-1} \) such that \( a'_{i,j} = a_{i,j} \) if \( i \geq j \) and otherwise \( a'_{i,j} = 0 \). The zero graph will be denoted by \( G_0 \) where for all \( i, j \in [n] \), \( a_{i,j} = 0 \).

Let \( \Sigma \) be a ring and \( G_1 \) and \( G_2 \) be two graphs over \( \Sigma \) with the same node set \( V \). The operator “+” between \( G_1 \) and \( G_2 \) over \( \Sigma \), is defined by \( G_1 + G_2 = G_3 \), where \( G_3 \) is the unique graph satisfying \( A_{G_1} + A_{G_2} = A_{G_3} \). Similarly, the operator “\( \cdot \)” between \( G_1 \) and an element \( \alpha \in \Sigma \), is denoted by \( \alpha \cdot G_1 = G_3 \), where \( G_3 \) is the unique graph satisfying \( \alpha \cdot A_{G_1} = A_{G_3} \).

A code over graphs over \( \Sigma \) of length \( n \) and size \( M \) is a set of graphs \( \mathcal{C} = \{ G_i = (V_n, L_i) | i \in [M] \} \) over \( \Sigma \), and it will be denoted by \((n,M)_{\Sigma}\). In case that \( \Sigma = \{0,1\} \), we simply use the notation \((n,M)\). The dimension of a code over graphs \( \mathcal{C} \) is \( k = \log_{|\Sigma|} M \) and the redundancy is \( r = \binom{n+1}{2} - k \). A code over graphs \( \mathcal{C} \) over a ring \( \Sigma \) will be called linear and will be denoted by \( U-[n,k]_{\Sigma} \) if for every \( G_1,G_2 \in \mathcal{C} \) and \( \alpha, \beta \in \Sigma \), \( \alpha G_1 + \beta G_2 \in \mathcal{C} \).

The neighborhood edge set of the \( i \)th node of an undirected graph \( G = (V_n, L) \) is defined by \( N_i = \{ \langle v_i, v_j \rangle | j \in [n] \} \), and it corresponds to the \( i \)th column and the \( i \)th row in the labeling matrix \( A_G \). The node failure of the \( i \)th node is the event in which all the edges in the neighborhood set of the \( i \)th node, i.e. \( N_i \), are erased. We will also denote this edge set by \( F_i \) and refer to it by the failure set of the \( i \)th node. A code over graphs is called a \( \rho \)-node-erasure-correcting code if it can correct any failure of at most \( \rho \) nodes in each of its graphs.

As discussed in \[23, 24, 25\], according to the Singleton bound, the minimum redundancy \( r \) of any \( \rho \)-node-erasure-correcting code of length \( n \), satisfies

\[
 r \geq \binom{n + 1}{2} - \binom{n - \rho + 1}{2} = n\rho - \binom{\rho}{2},
\]

and a code over graphs which satisfies this inequality with equality is called optimal. It was also observed in \[23, 24, 25\] that for all \( n \) and \( \rho \), an optimal \( \rho \)-node-erasure-correcting code exists over a field of size at least \( \Theta(n^2) \), and thus the goal is to construct such codes over smaller fields, and ideally over the binary field.

We conclude this section with reviewing the definition of a distance metric over graphs from \[25\] and its connection to the construction of codes correcting node failures. Let \( G = (V_n, L) \) be a graph and let \( E \) be the set of all nonzero labeled edges of \( G \), i.e., \( E = \{ e \in V_n \times V_n | L(e) \neq 0 \} \). A vertex cover \( W \) of \( G \) is a subset of \( V_n \) such that for each \( \langle v_i, v_j \rangle \in E \) either \( v_i \in W \) or \( v_j \in W \). The
The graph weight of $G$ is defined by

$$w(G) = \min_{W \text{ is a vertex cover of } G} \{|W|\},$$

and the graph distance between two graphs $G_1, G_2$ will be denoted by $d(G_1, G_2)$ where it holds that $d(G_1, G_2) = w(G_1 - G_2)$. It was proved in [25] that this graph distance is a metric. The minimum distance of a code over graphs $C$, denoted by $d(C)$, is the minimum graph distance between any two distinct graphs in $C$, that is

$$d(C) = \min_{G_1 \neq G_2} \{d(G_1, G_2)\},$$

and in case the code is linear $d(C) = \min_{G \in C, G \neq G_0} \{w(G)\}$. Lastly, we state the following theorem from [25] that establishes the connection between the graph distance and the node-erasure-correction capability.

**Theorem 22.** A linear code over graphs $C$ is a $\rho$-node-erasure-correcting code if and only if its minimum distance satisfies $d(C) \geq \rho + 1$.

Let $n \geq 2$ be a prime number. Denote by $\mathcal{R}_n$ the ring of polynomials of degree at most $n - 1$ over $\mathbb{F}_2$. It is well known that $\mathcal{R}_n$ is isomorphic to the ring of all polynomials in $\mathbb{F}_2$ modulo $x^n - 1$. Denote by $M_n(x) \in \mathcal{R}_n$ the polynomial $M_n(x) = \sum_{\ell=0}^{n-1} x^\ell$ over $\mathbb{F}_2$, where it holds that $M_n(x)(x+1) = x^n - 1$ as a multiplication of polynomials over $\mathbb{F}_2[x]$. To avoid confusion in the sequel, since we are using only polynomials over $\mathbb{F}_2$, the notation $x^\ell + 1$ for all $\ell \in [n]$, will refer to a polynomial in $\mathcal{R}_n$ and for $\ell = n$, we will use the notation $x^n - 1$. It is well known that for all $\ell \in [n]$ it holds that

$$\gcd(x^\ell + 1, x^n - 1) = x^{\gcd(\ell, n)} + 1 = x + 1,$$

and since $M_n(x)(x+1) = x^n - 1$ it can be verified that

$$\gcd(x^\ell + 1, M_n(x)) = 1. \quad (4.2)$$

Notice also that when 2 is primitive in $\mathbb{Z}_n$, the polynomial $M_n(x)$ is irreducible [3]. The last important and well known property we will use for polynomials over $\mathbb{F}_2$ is that for all $k = 2^j, j \in \mathbb{N}$ it holds that $1 + x^{sk} = (1 + x^s)^k$. The notation $\langle a \rangle_n$ will be used to denote the value of $(a \mod n)$.

### 4.3 Optimal Binary Double-Node-Erasure-Correcting Codes

In this section we present a family of optimal binary linear double-node-erasure-correcting codes with $n$ nodes, where $n$ is a prime number.

Remember that for $i \in [n]$ the $i$th neighborhood set of the $i$th node is $N_i = \{(v_i, v_j) \mid j \in [n]\}$. Let $n \geq 3$ be a prime number and let $G = (V_n, L)$ be a graph
with \( n \) vertices. For \( h \in [n] \) we define the neighborhood of the \( h \)th node without itself self-loop by

\[
S_h = \{ \langle v_h, v_\ell \rangle \mid \ell \in [n], h \neq \ell \}. \tag{4.3}
\]

We also define for \( m \in [n] \), the \( m \)th diagonal set by

\[
D_m = \{ \langle v_k, v_\ell \rangle | k, \ell \in [n], (k + \ell)_n = m \}. \tag{4.4}
\]

The sets \( S_h \) for \( h \in [n] \) will be used to represent parity constraints on the neighborhood of each node and similarly the sets \( D_m \) for \( m \in [n] \) will be used to represent parity constraints on the diagonals with slope one in the labeling matrix \( A_G \). We state that for all \( m \in [n] \), the size of \( D_m \) is \( \frac{n+1}{2} \). This holds since in each neighborhood \( N(v_i) \), there is only a single edge which belongs to \( D_m \), which is the edge \( \langle v_i, v_{(m-1)} \rangle \). Another important observation is that \( D_m \) contains only a single self-loop which is the edge \( \langle v_{(m-2)} \rangle \).

**Example 9.** In Fig. 4.1 we demonstrate the sets \( S_h \) and \( D_m \), where \( h, m \in [11] \), of a graph \( G = (V_{11}, L) \) on its lower-triangle-labeling matrix \( A'_G \).

Motivated by the algebraic approach of the work in [4, 5], and [11], we introduce one more useful notation for graphs. Let \( G = (V, L) \) be a graph. For \( i \in [n] \) we denote the *neighborhood-polynomials* of \( G \) to be

\[
a'_i(x) = e_{i,0} + e_{i,1}x + e_{i,2}x^2 + \cdots + e_{i,n-1}x^{n-1},
\]

where for \( i, j \in [n] \), \( e_{ij} = a_{ij} = L(v_i, v_j) \). We also denote the *neighborhood-polynomial without self-loops* of \( G \) to be

\[
a_i(x) = a'_i(x) - e_{i,i}x^i.
\]

We are now ready to present the construction of optimal double-node-erasure-correcting codes.
Construction 9. Let \( n \geq 3 \) be a prime number. The code over graphs \( C_2 \) is defined as follows,
\[
C_2 = \left\{ G = (V_n, L) \left| \begin{array}{l}
(a) \sum_{(v_i, v_j) \in S_h} e_{i,j} = 0, h \in [n] \\
(b) \sum_{(v_i, v_j) \in D_m} e_{i,j} = 0, m \in [n]
\end{array} \right. \right\}.
\]

Note that for any graph \( G \) over the binary field, it holds that
\[
\sum_{h \in [n]} \sum_{(v_i, v_j) \in S_h} e_{i,j} = \sum_{h=0}^{n-1} \sum_{\ell=0, \ell \neq h}^{n-1} e_{h,\ell} = 0.
\] (4.5)

Therefore the code \( C_2 \) has at most \( 2n - 1 \) linearly independent constraints which implies that its redundancy is at most \( 2n - 1 \). Our main result in this section, which is stated in Theorem 23, claims that \( C_2 \) is a double-node-correcting code, i.e. its minimum distance is three. Thus, according to the Singleton bound we get that the redundancy of the code \( C_2 \) is exactly \( 2n - 1 \), which implies that it is an optimal code. In the rest of this section we provide the proof Theorem 23 by showing a complexity optimal decoder for the code \( C_2 \) and prove its correctness.

Throughout this section we assume that \( G \) is a graph in the code \( C_2 \) and \( a_\ell(x) \) for \( \ell \in [n] \) are its neighborhood polynomials. We also assume that the failed nodes are \( v_0, v_i \). First, we define the following two polynomials \( S_1(x), S_2(x) \in \mathbb{R}_n \), which will be called the syndrome polynomials
\[
S_1(x) = a_0(x) + a_i(x),
\]
\[
S_2(x) = a_0(x) + a_i(x)x^i (\mod x^n - 1).
\]

Next, we prove the following claim.

Claim 10. The following properties hold on the graph \( G \):

1. For all \( h \in [n] \setminus \{0, i\} \), the value of \( e_{h,0} + e_{h,i} \) is known.
2. For all \( m \in [n] \setminus \{i\} \), the value of \( e_{0,m} + e_{i,(m-i)_n} \) is known.
3. The value of \( e_{0,i} \) is known.

Proof. 1. According to the neighborhood constraint \( S_h \) for all \( h \in [n] \setminus \{0, i\} \), we have that
\[
0 = \sum_{(v_i, v_j) \in S_h} e_{i,j} = \sum_{\ell=0, \ell \neq h}^{n-1} e_{h,\ell} = e_{h,0} + e_{h,i} + \sum_{\ell \in [n] \setminus \{0, i, h\}} e_{h,\ell},
\]
and since \( e_{h,\ell} \) are known for all \( \ell \in [n] \setminus \{0, i, h\} \), we get that the value of \( e_{h,0} + e_{h,i} \) is known.
2. For all \( m \in [n] \setminus \{i\} \), the set \( D_m \setminus \{ \langle v_0, v_m \rangle, \langle v_i, v_{(m-i)n} \rangle \} \) is denoted by \( D'_m \). Therefore, we have that
\[
0 = \sum_{\langle v_i, v_{(m-l)n} \rangle \in D_m} e_{\ell,(m-l)n} = \sum_{\langle v_i, v_{(m-l)n} \rangle \in D'_m} e_{\ell,(m-l)n} + e_{0,m} + e_{i,(m-i)n},
\]
and since the value of \( e_{s,\ell} \) is known for all \( \langle v_s, v_\ell \rangle \in D'_m \), we get that the value of \( e_{0,m} + e_{i,(m-i)n} \) is known.

3. According to the diagonal constraint \( D_i \) we get that
\[
0 = \sum_{\langle v_s, v_\ell \rangle \in D_i} e_{s,\ell} = e_{0,i} + \sum_{\langle v_s, v_\ell \rangle \in D_i \setminus \{ \langle v_0, v_i \rangle \}} e_{s,\ell},
\]
and since the value of \( e_{s,\ell} \) is known for all \( \langle v_s, v_\ell \rangle \in D_i \setminus \{ \langle v_0, v_i \rangle \} \), the value of \( e_{0,i} \) is known.

Denote the polynomials \( \tilde{S}_1(x) \) and \( \tilde{S}_2(x) \) by
\[
\begin{align*}
\tilde{S}_1(x) &= \sum_{h=1, h \neq i}^{n-1} (e_{0,h} + e_{i,h}) x^h, \\
\tilde{S}_2(x) &= \sum_{m=0, m \neq i}^{n-1} (e_{0,m} + e_{i,(m-i)n}) x^m.
\end{align*}
\]

Note that it is possible to compute \( \tilde{S}_1(x) \) and \( \tilde{S}_2(x) \) based upon the nodes that did not fail, and as we showed in Claim 10. Also note that calculating \( \tilde{S}_1(x) \) requires \((n-2)(n-4)\) XOR operations since in each of the \( n-2 \) constraints we read \( n-3 \) edges. Similarly, \( \tilde{S}_2(x) \) requires \((n-1)(n-5)/2\) XOR operations since in each of the \( n-1 \) constraints we read \( (n-3)/2 \) edges. Next we show an important claim.

Claim 11. The following properties hold on the graph \( G \):

1. For all \( h \in [n] \), \( a_h(1) = 0 \).

2. \( a_0(x) + a_i(x) = e_{i,0}(1 + x^i) + \tilde{S}_1(x) \).

3. \( a_0(x) + a_i(x)x^i \equiv e_{0,0} + e_{i,i}x^{2i} + \tilde{S}_2(x) \pmod{x^n - 1} \).

Proof. 1. By the definition of the neighborhood constraints, for all \( h \in [n] \),
\[
S_h = \{ \langle v_h, v_\ell \rangle \mid \ell \in [n], h \neq \ell \},
\]
and therefore
\[
a_h(1) = \sum_{\ell=0, \ell \neq h}^{n-1} e_{h,\ell} = \sum_{\langle v_h, v_\ell \rangle \in S_h} e_{h,\ell} = 0.
\]
2. 

\[ a_0(x) + a_i(x) = \]
\[ = e_{0,0} + e_{i,i}x^i + \sum_{h=0}^{n-1} e_{0,h}x^h + \sum_{h=0}^{n-1} e_{i,h}x^h \]
\[ = e_{0,0} + e_{i,i}x^i + \sum_{h=0}^{n-1} (e_{0,h} + e_{i,h})x^h \]
\[ = e_{0,0} + e_{i,i}x^i + (e_{0,0} + e_{i,0}) + (e_{0,i} + e_{i,i})x^i + \tilde{S}_1(x) \]
\[ = e_{i,0}(1 + x^i) + \tilde{S}_1(x). \]

3. 

\[ a_0(x) + a_i(x)x^i = \]
\[ = e_{0,0} + e_{i,i}x^{2i} + \sum_{m=0}^{n-1} e_{0,m}x^m + \sum_{m=0}^{n-1} e_{i,m}x^{m+i} \]
\[ \equiv e_{0,0} + e_{i,i}x^{2i} + \sum_{m=0}^{n-1} e_{0,m}x^m + \sum_{m=0}^{n-1} e_{i,(m-i)_n}x^m (\text{mod } x^n - 1) \]
\[ \equiv e_{0,0} + e_{i,i}x^{2i} + \sum_{m=0}^{n-1} (e_{0,m} + e_{i,(m-i)_n})x^m (\text{mod } x^n - 1) \]
\[ \equiv e_{0,0} + e_{i,i}x^{2i} + (e_{0,i} + e_{i,0})x^i + \tilde{S}_2(x) (\text{mod } x^n - 1) \]
\[ \equiv e_{0,0} + e_{i,i}x^{2i} + \tilde{S}_2(x) (\text{mod } x^n - 1). \]

If no nodes have failed in the graph \( G \), then we can easily compute both of these polynomials since we know the values of all the edges. However, in case that \( v_0, v_i \) both failed this becomes a far less trivial problem. However, using several properties, that will be proved in this section, we will prove that it is still possible to compute \( S_1(x) \) and \( S_2(x) \) entirely even though the nodes \( v_0, v_i \) failed. First, we show that it is possible to compute \( S_1(x) \) and almost compute \( S_2(x) \).

Claim 12. Given the two node failures \( v_0, v_i \), it is possible to exactly compute the polynomial \( S_1(x) \).

Proof. By using the result of Claim 11[2] we deduce that

\[ S_1(x) = a_0(x) + a_i(x) = e_{i,0}(1 + x^i) + \tilde{S}_1(x). \]

According to Claim 10[4] we can compute the polynomial \( \tilde{S}_1(x) \) and due to Claim 10[3] we can compute \( e_{i,0} \). Thus, we can compute the polynomial \( S_1(x) \).
According to Claim 10(3) it is possible compute the edge $e_{i,0}$ and this calculation requires $(n - 3)/2$ XOR operations. Since the calculation of $\tilde{S}_1(x)$ requires $(n - 2)(n - 4)$ XOR operations, we deduce that it takes $(n - 3)/2 + (n - 2)(n - 4)$ XOR operations to calculate $S_1(x)$. Next we show how to calculate the polynomial $S_2(x)$.

Claim 13. It is possible to compute all of the coefficients of the polynomial $S_2(x)$ except for the coefficients of $x^0$ and $x^{(2i)n}$.

Proof. By using the result of Claim 11(3) we deduce that

$$S_2(x) = a_0(x) + a_i(x)x^i$$

$$= e_{0,0} + e_{i,i}x^i + \tilde{S}_2(x)(\mod x^n - 1).$$

The polynomial $\tilde{S}_2(x)$ can be computed due to Claim 10(2). The only coefficients in this polynomial that we can not compute are $x^0$ and $x^{(2i)n}$, which are dependent on the edges $e_{0,0}$ and $e_{i,i}$. □

After we compute $e_{0,0}$ and $e_{i,i}$, computing $S_2(x)$ requires the same number of XOR operations as we did for $\tilde{S}_2(x)$ which is $(n - 1)(n - 5)/2$. We now show how to compute $a_0(x)$ and $a_i(x)$.

Claim 14. Given the values of $e_{0,0}, e_{i,i}$, we can compute the polynomials $a_0(x)$ and $a_i(x)$, i.e., decode the failed nodes $v_0, v_i$.

Proof. Assume that the values of $e_{0,0}, e_{i,i}$ are known. This implies that we can compute exactly the polynomials $S_1(x)$ as well as $S_2(x)$ and let us denote

$$S_1(x) + S_2(x) \equiv \sum_{k=0}^{n-1} s_k x^k (\mod x^n - 1),$$

that is, the coefficients $s_k$ for $k \in [n]$ are known. By the definition of $S_1(x)$ and $S_2(x)$ we have that

$$S_1(x) = a_0(x) + a_i(x),$$

$$S_2(x) \equiv a_0(x) + a_i(x)x^i (\mod x^n - 1).$$

Adding up these two equations results with

$$S_1(x) + S_2(x) \equiv a_i(x) + a_i(x)x^i (\mod x^n - 1).$$

Thus, we get the following $n$ equations with the $n$ variables $e_{i,k}$ for $k \in [n]$. For all $k \in [n] \setminus \{i, (2i)\}$ we get the equation

$$e_{i,k} + e_{i,(k-i)n} = s_k, \quad (4.8)$$

80
for \( k = i \) we get the equation 

\[ e_{i,0} = s_i, \]

and lastly for \( k = \langle 2i \rangle \) we get the equation 

\[ e_{i,\langle 2i \rangle} = s_{\langle 2i \rangle}. \]

Since \( e_{i,0} \) and \( s_k \) for \( k \in [n] \) are known and since \( n \) is a prime number, by a simple induction using (4.8) the edges \( e_{i,\langle -i \rangle}, e_{i,\langle -2i \rangle}, \ldots, e_{i,\langle 4i \rangle}, e_{i,\langle 3i \rangle} \) can be calculated. Hence, at this point all of the coefficients of \( a_i(x) \) are known besides \( e_{i,i} \), which assures the claim’s statement for \( a_i(x) \). Lastly, the polynomial \( a_0(x) \) can be decoded by the value of \( e_{0,0} \) and the equality \( a_0(x) = S_1(x) + a_i(x) \).

An important observation is that calculating \( S_1(x) + S_2(x) \) requires \( n \) XOR operations. After that, calculating \( a_i(x) \) requires \( n - 3 \) XORs due to (4.8). The polynomial \( a_0(x) \) requires \( n \) more XORs by \( a_0(x) = S_1(x) + a_i(x) \). We will show that \( e_{0,0} \) and \( e_{i,i} \) requires \( 5(n - 3) \) XORs and so, computing \( a_0(x) \) and \( a_i(x) \) requires

\[
(n - 2)(n - 4) + (n - 3)/2 + (n - 1)(n - 5)/2 + 5(n - 3)
\]

\[+ 2n + n - 3 = 3 \frac{n^2}{2} - \frac{1}{2} n - 9\]

XOR operations.

Now all that is left to show is the decoding of \( e_{0,0}, e_{i,i} \). This will be done in two steps; first we will decode the values of \( e_{i,n-i}, e_{0,\langle 2i \rangle} \) and then we will derive the values of \( e_{0,0}, e_{i,i} \). The former edges will be decoded using the following algorithm.

**Algorithm 3** Decoding of \( e_{0,\langle 2i \rangle} \)

1: Decode \( e_{0,i} \) using the \( D_i \) constraint
2: \( \ell = 3 \)
3: \( \text{sum} = e_{0,i} \)
4: while \( \ell < n - 1 \) do
5: \( \text{Compute } d_{\ell} = e_{0,\langle \ell \cdot i \rangle} + e_{i,\langle \ell \cdot i \rangle} \)
6: \( \text{Compute } f_{\ell} = e_{i,\langle \ell \cdot i \rangle} + e_{0,\langle (\ell + 1) \cdot i \rangle} \)
7: \( \text{sum} = \text{sum} + d_{\ell} + f_{\ell} \)
8: \( \ell = \ell + 2 \)
9: end while
10: \( e_{0,\langle 2i \rangle} = \text{sum} \)

Using a similar algorithm we decode the value \( e_{i,n-i} \) as well. To prove the correctness of Algorithm 3 it suffices that we prove the following claim.

**Claim 15.** All steps in Algorithm 3 are possible to compute and furthermore, \( \text{sum} = e_{0,\langle 2i \rangle} \).
**Proof.** First we compute the edge $e_{0,i}$ due to Claim 10. Next, the values $\ell$ receives in the while loop of the algorithm are $3, 5, \ldots, n-2$ and for every value of $\ell$ it is possible to compute $d_{\ell}$ by the neighborhood constraint of $S_{\langle \ell i \rangle}$. Similarly, the value of $f_{\ell}$ is computed by the diagonal constraint $D_{\langle (\ell + 1) i \rangle}$.

From the while loop of Algorithm 3 we have that

$$ \text{sum} = e_{0,i} + \sum_{k=1}^{n-3} (d_{2k+1} + f_{2k+1}) $$

$$ = e_{0,i} + \sum_{k=1}^{n-3} (e_{0,\langle (2k+1) i \rangle} + e_{0,\langle (2k+2) i \rangle}) $$

$$ = \sum_{\ell=1, \ell \neq 2}^{n-1} e_{0,\langle \ell i \rangle} (a) = \sum_{\ell=1, \ell \neq 2}^{n-1} e_{0,\ell} (b) = e_{0,\langle 2i \rangle} - e_{0,i}. $$

Step (a) holds since $i$ is a generator of the group $\mathbb{Z}_n$, and thus

$$ \{ \langle 3i \rangle, \langle 4i \rangle, \ldots, \langle (n-1) \cdot i \rangle \} $$

are all distinct elements in $\mathbb{Z}_n$, and since we also added the term $e_{0,i}$ to this summation. Lastly, Step (b) holds by the neighborhood constraint of $S_0$ and we get that $\text{sum} = e_{0,\langle 2i \rangle}$.

Given the value of $e_{i,0}$, computing the edge $e_{0,\langle 2i \rangle}$ in Algorithm 3 requires $2(n-3)$ XORs. Next, the edge $e_{i,i}$ is calculated using the diagonal constraint $D_{\langle 2i \rangle}$, which requires $(n-3)/2$ XORs. The edge $e_{0,0}$ us calculated in the same manner and requires $5(n-3)/2$ XOR operations.

To summarize, given the values of $e_{i,i}, e_{0,0}$, an efficient decoding procedure with time complexity $\Theta(n^2)$ works as follows:

**Algorithm 4**

1: Compute $S_1(x), S_2(x)$
2: Compute $S_1(x) + S_2(x)$
3: Solve the linear system of equations induced from the equality

$$ S_1(x) + S_2(x) \equiv a_i(x) + a_i(x)x^i (\mod (x^n - 1)) $$

in order to decode $a_i(x)$
4: Use the equality $a_0(x) = S_1(x) + a_i(x)$ in order to decode $a_0(x)$

Finally, using the properties above we conclude that it is possible to decode the polynomials $a_0(x)$ and $a_i(x)$ using $\frac{3}{2}n^2 - \frac{1}{2}n - 9$ XOR operations and the following theorem is established. Note that the decoding complexity is optimal since the input size is $\Theta(n^2)$.

**Theorem 23.** The decoding Algorithm 4 to the code $C_2$, efficiently corrects any two node failures. Its complexity is $\Theta(n^2)$, where $n$ is the number of nodes.
4.4 Single Node Regeneration

In this section, we follow the recent works on regenerating codes \cite{7, 14, 18} in order to analyze a sufficient number of edges we have to read in order to correct a single node erasure while using the code $C_2$. Our goal is to show that in order to correct a single node erasure, we are not required to read the rest of the graph in its entirety. Namely, while the number of edges in the graph is $\frac{n(n+1)}{2}$, we show that it is enough to read only $\frac{5}{12}n^2 + O(n)$ edges in order to decode a single node failure. The main result of this section is summarized in the following theorem.

**Theorem 24.** For any graph $G \in C_2$ with a single node failure, it suffices to read $\frac{5}{12}n^2 + O(n)$ edges in order to correct the node failure.

From the symmetry of the code, the algorithm can assume that $v_0$ is the failed node and it is decoded as follows. The $x$ edges $e_{0,n-1}, \ldots, e_{0,n-x}$ will be corrected using the neighborhood constraints $S_i, i \in [n]$. The rest of the edges will be corrected using the diagonal constraints $D_i, i \in [n]$. Throughout the section, to simplify the calculations, we assume that when we read the sets $S_i$ for $i \in [n]$, we also read the self loop edge $e_{i,i}$, that is, we read the neighborhood edge set $N_i = S_i \cup \{e_{i,i}\}$. We also assume that the edges of the node $v_0$ are read at the decoding algorithm as well. Note that these two assumptions can only weaken the result on the number of edges that are read in order to decode the node $v_0$ and they do not affect the statement in Theorem \ref{thm:main}. Let $R(x)$ be the set of edges that are read in order to correct the $x$ edges $e_{0,n-1}, \ldots, e_{0,n-x}$, that is, $R(x) = \bigcup_{1 \leq i \leq x} N_{n-i}$. We begin with the following claim.

**Claim 16.** Let $k \in [n]$. For all $A \subseteq [n]$, such that $|A| = k$, it holds that $|\bigcup_{t \in A} N_t| = nk - \binom{k}{2}$.

**Proof.** Clearly, $\sum_{t \in A} |N_t| = nk$, and for all distinct $t_1, t_2 \in [n]$ it holds that $|N_{t_1} \cap N_{t_2}| = 1$. Note that if $k \geq 3$, for all $B \subseteq [n]$, such that $3 \leq |B| \leq k$, it holds that $|\bigcap_{t \in B} N_t| = 0$.

Thus, we deduce that

$$|\bigcup_{t \in A} N_t| = \sum_{t \in A} |N_t| - \sum_{t_1, t_2 \in A, t_1 \neq t_2} |N_{t_1} \cap N_{t_2}| = nk - \binom{k}{2}.$$ 

By Claim \ref{claim:intersection}, we immediately deduce that $|R(x)| = nx - \binom{x}{2}$.

**Example 10.** Fig. 10 demonstrates the edges that are read for the case of $n = 19, x = 6$ when the failed node is $v_0$. The red part is the edges which are read.
in order to decode the edges $e_{0,13}, e_{0,14}, \ldots, e_{0,18}$ by the neighborhood constraints $S_i, i \in [19] \setminus [13]$. The green and blue parts are additional edges which are read in order to decode the edges $e_{0,0}, e_{0,1}, \ldots, e_{0,12}$ by the diagonal constraints $D_i, i \in [13]$. 

![Figure 4.2: Demonstration for rebuilding ratio calculation for single node failure](image)

Next, we prove the following claim where we assume that $x \leq \lfloor n/2 \rfloor$. For convenience, for all $i \in [n-x]$, we denote the set $A_i(x) = \{e_{n-z,i+z} | 1 \leq z \leq x\}$. Remember that $D_i = \{e_{k,\ell} | k, \ell \in [n], (k + \ell) \equiv n \equiv i \}$. 

**Claim 17.** The following properties hold:

1. For all $i \in [n-x]$, $D_i \cap R(x) = A_i(x)$.
2. For all $i \in [n-2x]$ and $1 \leq z \leq x$, $n-z > i+z$.
3. For all $i \in [n-2x]$, $|A_i(x)| = x$.
4. For all $i \in [n-x] \setminus [n-2x]$ and $1 \leq z < \left\lfloor \frac{n-i}{2} \right\rfloor$, $n-z > i+z$.
5. For all $i \in [n-x] \setminus [n-2x]$, $|A_i(x)| = \left\lfloor \frac{n-i}{2} \right\rfloor$.

**Proof.**

1. For all $1 \leq z \leq x$ it holds that $e_{n-z,i+z} \in N_{n-z}$ and thus $A_i(x) \subseteq R(x)$. Clearly, $e_{n-z,i+z} \in D_i$. Hence, $A_i(x) \subseteq D_i \cap R(x)$ and it is possible to verify the other direction, so $A_i(x) = D_i \cap R(x)$.

2. It holds that

$$n-z \geq n-x > i+x \geq i+z,$$

where (a) and (c) hold since $1 \leq z \leq x$ and (b) holds since $i \in [n-2x]$.

3. By the definition of $A_i(x)$, for all $i \in [n-2x]$, $|A_i(x)| \leq x$. By (2), for all $i \in [n-2x]$ and $1 \leq z \leq x$, $n-z > i+z$. Therefore, every value of $z$ between 1 and $x$ generates a unique edge $e_{n-z,i+z}$. Thus, by the definition of $A_i(x)$, we deduce that $|A_i(x)| = x$. 

84
4. It holds that
\[ n - z > n - \left\lceil \frac{n - i}{2} \right\rceil \geq i + \left\lceil \frac{n - i}{2} \right\rceil > i + z, \]
where (a) and (c) hold since \(1 \leq z < \left\lceil \frac{n - i}{2} \right\rceil\) and (b) holds since \(n - 2 \left\lceil \frac{n - i}{2} \right\rceil \geq i.

5. By (4), for all \(i \in [n - x] \setminus [n - 2x]\) and \(1 \leq z < \left\lfloor \frac{n - i}{2} \right\rfloor\), \(n - z > i + z\).

Therefore, every value of \(z\) between 1 and \(\left\lfloor \frac{n - i}{2} \right\rfloor - 1\) generates a unique edge \(e_{n-z,i+z}\). Moreover, for \(z = \left\lfloor \frac{n - i}{2} \right\rfloor\), \(n - z \geq i + z\). Thus, \(|A_i(x)| \geq \left\lfloor \frac{n - i}{2} \right\rfloor\).

Claim 18. It holds that
\[
\sum_{i=n-2x}^{n-x-1} |A_i(x)| = x^2 - \sum_{i=0}^{x-1} \left\lfloor \frac{i}{2} \right\rfloor.
\]

Proof. Note that,
\[
\sum_{i=n-2x}^{n-x-1} |A_i(x)| = \sum_{i=n-2x}^{n-x-1} \left\lfloor \frac{n - i}{2} \right\rfloor \\
= \sum_{i=0}^{x-1} \left\lfloor \frac{n - i - n + 2x}{2} \right\rfloor = \sum_{i=0}^{x-1} \left(x - \left\lfloor \frac{i}{2} \right\rfloor \right) \\
= x^2 - \sum_{i=0}^{x-1} \left\lfloor \frac{i}{2} \right\rfloor.
\]

Denote by \(F(x)\) the number of edges that we have to read in order to reconstruct all the edges in \(N_0\). Now we are ready to present the proof of Theorem 24.

Proof of Theorem 24. Note that the value \(|R(x)|\) corresponds to the number of edges that are read to decode the \(x\) edges \(e_{0,n-x}, \ldots, e_{0,n-1}\), and \(\sum_{i=0}^{n-x-1} (|D_i| - |A_i(x)|)\) is the remaining number of edges in order to decode the first \(n - x\) edges.
Thus,
\[ F(x) = |R(x)| + \sum_{i=0}^{n-x-1} \left( |D_i| - |A_i(x)| \right) \]
\[ = |R(x)| + \sum_{i=0}^{n-x-1} |D_i| - \sum_{i=0}^{n-2x-1} |A_i(x)| - \sum_{i=n-2x}^{n-x-1} |A_i(x)| \]
\[ = n x - \left( \frac{x}{2} \right) + (n-x) \frac{n+1}{2} - (n-2x)x \]
\[ - \left( x^2 - \sum_{i=0}^{x-1} \left\lfloor \frac{i}{2} \right\rfloor \right). \]

In the last equation, Step (a) holds since by Claim 16, \( |R(x)| = nx - \left( \frac{x}{2} \right) \), using the fact that for all \( i \in [n], |D_i| = \frac{n+1}{2} \), also by Claim 17, for all \( i \in [n-2x], |A_i(x)| = x \), and also using Claim 18 in which
\[ \sum_{i=n-2x}^{n-x-1} |A_i(x)| = x^2 - \sum_{i=0}^{x-1} \left\lfloor \frac{i}{2} \right\rfloor. \]

Lastly, it is possible to check that applying \( x = \left\lceil \frac{n}{3} \right\rceil \) provides that \( F\left( \left\lceil \frac{n}{3} \right\rceil \right) \leq \frac{5}{12}n^2 + \frac{n}{2} = \frac{5}{12}n^2 + O(n) \) as required. Thus it is sufficient to read only \( \frac{5}{12}n^2 + O(n) \) edges in this decoding algorithm for \( v_0 \). This concludes the proof of Theorem 24.

### 4.5 Binary Triple-Node-Erasure-Correcting Codes

In this section we present a construction of binary triple-node-erasure-correcting codes for undirected graphs. Let \( n \geq 5 \) be a prime number such that 2 is a primitive number in \( \mathbb{Z}_n \). Let \( G = (V_n, L) \) be a graph with \( n \) vertices. We will use in this construction the edge sets \( S_h, D_m \) for \( h \in [n], m \in [n] \) which were defined in (4.3), (4.4), respectively. In addition, for \( s \in [n] \) we define the edge set
\[ T_s = \{ \langle v_k, v_\ell \rangle | k, \ell \in [n], (k + 2\ell)_n = s, k \neq \ell \}. \]

In this construction we impose the same constraints from Construction 9, that is, the sets \( S_h \) will be used to represent parity constraints on the neighborhood of each node, the sets \( D_m \) will represent parity constraints on the diagonals with slope one of \( A_G \), and furthermore the sets \( T_s \) will represent parity constraints on the diagonals with slope two of \( A_G \).

**Example 11.** In Fig. 4.3 we present the sets \( T_s, s \in [11] \) of a graph \( G = (V_{11}, L) \) on its labeling matrix \( A_G \), and its lower-triangle-labeling matrix \( A'_G \).

We are now ready to show the following construction.
Figure 4.3: The slope-two-diagonal constraints over undirected graphs, represented on the labeling matrix and the lower-triangle-labeling matrix.

Construction 10. For all prime number \( n \geq 5 \) where 2 is primitive in \( \mathbb{Z}_n \), let \( \mathcal{C}_3 \) be the following code:

\[
\mathcal{C}_3 = \left\{ G = (V_n, L) \mid \left\{ \begin{array}{l}
(a) \sum_{(v_i,v_j) \in S_h} e_{i,j} = 0, h \in [n] \\
(b) \sum_{(v_i,v_j) \in D_m} e_{i,j} = 0, m \in [n] \\
(c) \sum_{(v_i,v_j) \in T_s} e_{i,j} = 0, s \in [n] 
\end{array} \right. \right\}.
\]

Note that the code \( \mathcal{C}_3 \) is a subcode of the code \( \mathcal{C}_2 \) and for any graph \( G \) over the binary field, by (4.5) there are only \( n - 1 \) independent constraints (a) in Construction 10 and by the same principle,

\[
\sum_{s \in [n]} \sum_{(v_i,v_j) \in T_s} e_{i,j} = \sum_{s=0}^{n-1} \sum_{\ell=0}^{n-1} e_{(s-2\ell),n-\ell} = 2 \sum_{h=0}^{n-1} \sum_{\ell=0}^{n-1} e_{h,\ell} = 0. \tag{4.9}
\]

Therefore the code \( \mathcal{C}_3 \) has at most \( 3n - 2 \) linearly independent constraints which implies that its redundancy is not greater than \( 3n - 2 \). Since we will prove in Theorem 25 that \( \mathcal{C}_3 \) is a triple-node-correcting code, according to the Singleton bound we get that the code redundancy is at most a single bit away from optimality. Our main result in this section is showing the following theorem.

Theorem 25. For all prime number \( n \geq 5 \) such that 2 is primitive in \( \mathbb{Z}_n \), the code \( \mathcal{C}_3 \) is a triple-node-erasure-correcting code. It is at most a single bit away from optimality.

Proof. Assume on the contrary that there is a graph \( G = (V_n, L) \in \mathcal{C}_3 \) where \( w(G) \leq 3 \). We prove here only the case that \( w(G) = 3 \) since the case of \( w(G) \leq 2 \) holds according to Theorem 23. By the symmetry of Construction 10 it is sufficient to assume that a vertex cover \( W \) of \( G \) is \( W = \{v_0,v_i,v_j\} \) for distinct \( i,j \in [n] \setminus \{0\} \), while all other cases hold by relabeling the indices 0, i, j. We will show that \( G = G_0 \).
Denote by $H_{i,j}$ the set
\[ H_{i,j} = \{ i, j, (2i)_n, (2j)_n, (2i + j)_n, (2j + i)_n \}, \tag{4.10} \]
and for all $s \in [n]$, denote by $h_{i,j}(s)$ the sum
\[ h_{i,j}(s) = e_{0,s} + e_{i,(s-2i)}_n + e_{j,(s-2j)}_n \]
\[ + e_{0,(2-1)s}_n + e_{i,(2-1(s-i))}_n + e_{j,(2-1(s-j))}_n. \tag{4.11} \]

The next claim presents several useful properties.

**Claim 19.** The following properties hold on the graph $G$:

(a) For all $h \in [n] \setminus \{0, i, j\}$, $e_{0,h} + e_{i,h} + e_{j,h} = 0$.

(b) For all $m \in [n] \setminus \{i, j, (i+j)\}$, $e_{0,m} + e_{i,(m-i)}_n + e_{j,(m-j)}_n = 0$.

(c) $e_{0,i} + e_{j,(i-j)}_n = e_{0,j} + e_{i,(j-i)}_n = e_{j,i} + e_{0,(i+j)}_n = 0$.

(d) For all $s \in [n] \setminus H_{i,j}$, it holds that $h_{i,j}(s) = 0$.

(e) It holds that
\[
\sum_{s \in H_{i,j}} h_{i,j}(s)x^s \\
\equiv e_{i,0}(x^i + x^{2i}) + e_{j,0}(x^j + x^{2j}) + e_{j,i}(x^{2i+j} + x^{i+2j}) \pmod{x^n - 1}.
\]

**Proof.** We remind that for all $k, \ell \in [n] \setminus \{0, i, j\}$, $e_{k,\ell} = 0$.

(a) We know that for all $h \in [n] \setminus \{0, i, j\}, s \in [n] \setminus \{h\}$, $\langle v_s, v_h \rangle \in S_h$, and therefore by the definition of the constraint (a) in Construction 10 we get that
\[
0 = \sum_{\langle v_s, v_h \rangle \in S_h} e_{s,h} = \sum_{s=0, s \neq h}^{n-1} e_{s,h} = e_{0,h} + e_{i,h} + e_{j,h}.
\]

(b) For all $m \in [n] \setminus \{i, j, i+j\}$, denote by $D'_m$ the set
\[
D'_m = D_m \setminus \{ \langle v_0, v_m \rangle, \langle v_i, v_{(m-i)} \rangle, \langle v_j, v_{(m-j)} \rangle \}.
\]
Therefore, we have that
\[
0 = \sum_{\langle v_j, v_{(m-j)} \rangle \in D_m} e_{j,(m-j)}_n = \sum_{\langle v_j, v_{(m-j)} \rangle \in D'_m} e_{j,(m-j)}_n + e_{0,m} + e_{i,(m-i)}_n + e_{j,(m-j)}_n,
\]
and since $e_{s,k} = 0$ for all $\langle v_s, v_k \rangle \in D'_m$, we get that $e_{0,m} + e_{i,(m-i)}_n + e_{i,(m-i)}_n = 0$. 

88
(c) Similarly to (b), for \( m = i \) we get that \( \langle v_0, v_m \rangle = \langle v_i, v_{(m-i)} \rangle \) and therefore by the definition of the constraint (b) in Construction 10 we get that \( e_{0,i} + e_{i,(i-j)} = 0 \). It can be similarly verified that for \( m = j \) we get that \( e_{0,j} + e_{i,(j-i)} = 0 \) and for \( m = (i+j) \) we get that \( e_{j,i} + e_{0,(i+j)} = 0 \).

(d) For all \( s \in [n] \), let \( B_s \) be the following edge set

\[
B_s = \{ \langle v_0, v_s \rangle, \langle v_i, v_{(s-2i)} \rangle, \langle v_j, v_{(s-2j)} \rangle, \langle v_0, v_{(2i)} \rangle, \langle v_i, v_{(2i-1)} \rangle, \langle v_j, v_{(2i-1)} \rangle \}.
\]

(4.12)

It can be readily verified that for \( s \not\in \{0, (3i) \} \cup H_{i,j}, |B_s| = 6 \). For all \( k \in \{0, i, j\} \) and all \( s \in [n] \setminus \{0, (3i), (3j)\} \) it holds that \( k \neq s - 2k \) and therefore, if \( \langle v_k, v_{(s-2k)} \rangle \in B_s \) then \( \langle v_k, v_{(s-2k)} \rangle \in T_s \), i.e., \( B_s \subseteq T_s \). Therefore, by the definition of the diagonal constraint (c) in Construction 10 we deduce that for all \( s \not\in \{0, (3i), (3j)\} \cup H_{i,j}, \)

\[
0 = \sum_{(v_k, v_m) \in T_s} e_{k,m} = \sum_{(v_k, v_m) \in B_s} e_{k,m} = h_{i,j}(s).
\]

Moreover, for \( s = 0 \), \( \langle v_0, v_s \rangle = \langle v_0, v_{(2i)} \rangle = \langle v_0, v_0 \rangle \) and therefore \( |B_0| = 5 \). It can be similarly verified that \( |B_{(3i)}| = |B_{(3j)}| = 5 \). Notice that for all \( k \in \{0, i, j\}, s \in \{0, (3i), (3j)\} \), if \( \langle v_k, v_{(s-2k)} \rangle \in B_s \) then it holds that \( \langle v_k, v_{(s-2k)} \rangle \in T_s \cup \{\langle v_k, v_k \rangle\} \), i.e., \( B_s \subseteq T_s \cup \{\langle v_k, v_k \rangle\} \). Therefore again, by the definition of the diagonal constraint (c) in Construction 10 we deduce that for all \( s \in \{0, (3i), (3j)\} \),

\[
0 = \sum_{(v_k, v_m) \in T_s \cup \{\langle v_{(3i-s)} \rangle \cup \langle v_{(3j-s)} \rangle \}} e_{k,m} + e_{(3s-s)} \]

\[
= \sum_{(v_k, v_m) \in B_s} e_{k,m} + e_{(3s-s)} = h_{i,j}(s).
\]

(e) For all \( s \in H_{i,j} \) let \( B_s \) be the edge set from (4.12). Notice that for \( s = i \) we get that \( \langle v_0, v_s \rangle = \langle v_i, v_{(2i-s)} \rangle \), for \( s = (2i) \) we get that \( \langle v_i, v_{(s-2i)} \rangle = \langle v_0, v_{(2i)} \rangle \), and for \( s = (2i+j) \) we get that \( \langle v_i, v_{(s-2i)} \rangle = \langle v_j, v_{(2i+j-s)} \rangle \), and therefore for all \( s \in H_{i,j}, |B_s| = 5 \). Similarly to the proof of (d), the edge set \( B_s \) consists of all the edges incident to at least one of the nodes \( v_0, v_i \) and \( v_j \) in \( T_s \), i.e., \( B_s \subseteq T_s \). Therefore we deduce that for \( s \in \{i, j\}, \)

\[
e_{s0} = \sum_{(v_k, v_m) \in T_s} e_{k,m} + e_{s0} = \sum_{(v_k, v_m) \in B_s} e_{k,m} + e_{s0} = h_{i,j}(s),
\]

and the coefficient of the monomial \( x^i, x^j \) in the polynomial \( \sum_{s \in H_{i,j}} h_{i,j}(s) x^s \) is \( e_{i0}, e_{j0} \), respectively. The proof that the coefficient of \( x^{2i}, x^{2j}, x^{2i+j}, x^{2j+i} \) in this polynomial is \( e_{i,0}, e_{j,0}, e_{i,j}, e_{j,i} \) is similar, respectively.
From this claim we deduce the following equations.

\[
\sum_{h=1, h \notin \{i, j\}}^{n-1} (e_{0,h} + e_{i,h} + e_{j,h}) x^h = 0, \quad (4.13)
\]

\[
\sum_{m=0, m \notin \{i,j,(i+j)\}}^{n-1} (e_{0,m} + e_{i,(m-i)} + e_{j,(m-j)}) x^m = 0, \quad (4.14)
\]

\[
\sum_{s=0, s \notin H_{i,j}}^{n-1} h_{i,j}(s) x^s = 0. \quad (4.15)
\]

Next, let \(a_0(x), a_i(x)\) and \(a_j(x)\) be the neighborhood polynomials without self-loops of \(G\). The following lemma presents a few equalities that will be used to decode the values of \(a_0(x), a_i(x)\) and \(a_j(x)\).

**Lemma 26.** The following properties hold:

(a) \(a_0(x) + a_i(x) + a_j(x) = e_{i,0}(1 + x^i) + e_{j,0}(1 + x^j) + e_{j,i}(x^i + x^j).\)

(b) \(a_0(x) + a_i(x)x^i + a_j(x)x^j \equiv e_{i,0} + e_{i,x^2} + e_{j,j}x^{2j} + e_{i,0}x^i + e_{j,0}x^j + e_{j,i}x^{i+j} \pmod{x^n - 1}.\)

(c) \(a_0(x) + a_i(x)x^{2i} + a_j(x)x^{2j} + a_0^2(x) + a_i^2(x)x^i + a_j^2(x)x^j \equiv e_{i,0}(x^i + x^{2i}) + e_{j,0}(x^j + x^{2j}) + e_{j,i}(x^{2i+j} + x^{i+2j}) \pmod{x^n - 1}.\)

**Proof.** (a) According to the neighborhood-polynomials definition we can write

\[
a_0(x) + a_i(x) + a_j(x) = \sum_{h \in \{0,i,j\}} e_{h,h}x^h + \sum_{h=0}^{n-1} e_{0,h}x^h + \sum_{h=0}^{n-1} e_{i,h}x^h + \sum_{h=0}^{n-1} e_{j,h}x^h
\]

\[
= \sum_{h \in \{0,i,j\}} e_{h,h}x^h + \sum_{h=0}^{n-1} \left( e_{0,h} + e_{i,h} + e_{j,h} \right) x^h
\]

\[
= e_{0,0} + e_{i,i}x^i + e_{j,j}x^j + \left( e_{0,0} + e_{i,0} + e_{j,0} \right)
\]

\[
+ \left( e_{0,j} + e_{i,i} + e_{j,i} \right) x^i + \left(e_{0,j} + e_{i,j} + e_{j,j} \right) x^j
\]

\[
+ \sum_{h=1, h \notin \{i,j\}}^{n-1} \left( e_{0,h} + e_{i,h} + e_{j,h} \right) x^h
\]

\[
\equiv e_{i,0}(1 + x^i) + e_{j,0}(1 + x^j) + e_{j,i}(x^i + x^j),
\]
where Step (a) holds due to (4.13).

(b)

\[ a_0(x) + a_i(x)x^i + a_j(x)x^j = \]
\[ = \sum_{m \in \{0, i, j\}} e_{m,m}x^{2m} + \sum_{m=0}^{n-1} e_{0,m}x^m + \sum_{m=0}^{n-1} e_{i,m}x^{m+i} \]
\[ + \sum_{m=0}^{n-1} e_{j,m}x^{m+j} = \sum_{m \in \{0, i, j\}} e_{m,m}x^{2m} + \sum_{m=0}^{n-1} e_{0,m}x^m \]
\[ + \sum_{m=0}^{n-1} e_{i,\langle m-i \rangle_n}x^m + \sum_{m=0}^{n-1} e_{j,\langle m-j \rangle_n}x^m \mod (x^n - 1) \]
\[ = \sum_{m \in \{0, i, j\}} e_{m,m}x^{2m} + \sum_{m=0}^{n-1} \left( e_{0,m} + e_{i,\langle m-i \rangle_n} + e_{j,\langle m-j \rangle_n} \right)x^m \]
\[ = e_{0,0} + e_{i,i}x^2i + e_{j,j}x^2j \]
\[ + \left( e_{0,i} + e_{i,0} + e_{j,\langle i-j \rangle_n} \right)x^i + \left( e_{0,j} + e_{i,\langle j-i \rangle_n} + e_{j,0} \right)x^j \]
\[ + \left( e_{0,\langle i+j \rangle_n} + e_{i,j} + e_{j,i} \right)x^{i+j} \]
\[ + \sum_{m=0, m \notin \{i,j,\langle i+j \rangle_n\}}^{n-1} \left( e_{0,m} + e_{i,\langle m-i \rangle_n} + e_{j,\langle m-j \rangle_n} \right)x^m \]
\[ \overset{(a)}{=} e_{0,0} + e_{i,i}x^{2i} + e_{j,j}x^{2j} + e_{i,0}x^i + e_{j,0}x^j + e_{j,i}x^{i+j}, \]

where Step (a) holds due to (4.14).
Lemma 27. This lemma is given in Appendix A.

Lemma 28. It holds that \( e_{0,0} + e_{i,j} + e_{j,i} = e_{j,0} + e_{j,0} + e_{j,i} = 0 \).
By Lemma 28, we know that at least one of the self-loops $e_{j,j}$, $e_{i,i}$ or $e_{0,0}$ is zero, and our next step is showing that one of the polynomials $a_0(x), a_i(x)$ or $a_j(x)$ is zero. We assume that $e_{j,j}$ is zero, while the proof of the other two cases will be similar based upon Lemma 27b and 27c. By Lemma 27a, we get that

$$a_1 + p \equiv 0 \pmod{n}.$$ 

deduce that

$$p \equiv 0 \pmod{n}.$$ 

Notice that in this case, by Lemma 28 we have that $e = 0$ and therefore $a_i = e_{i,0} = 0$. By Lemma 21, we deduce that

$$a_0(x) + a_i(x) + 1 + x^i$$

$$= a_0(x) + a_i(x) + a_j(x)$$

$$= e_{i,0}(1 + x^i) + e_{j,0}(1 + x^j) + e_{j,i}(x^i + x^j)$$

$$= (1 + x^j) + (x^i + x^j) = 1 + x^i,$$

and therefore $a_0(x) + a_i(x) = 0$. Again, by Lemma 28 we know that $e_{0,0} + e_{i,i} + e_{j,j} = 0$ and therefore, since $e_{j,j} = 0$, we get that $e_{i,i} = e_{0,0}$. By Lemma 21, we deduce that

$$a_0(x) + a_i(x)x^i + (1 + x^i)x^j$$

$$= a_0(x) + a_i(x)x^i + a_j(x)x^j$$

$$\equiv e_{0,0} + e_{i,i}x^{2i} + e_{j,j}x^{2j} + e_{i,0}x^i + e_{j,0}x^j + e_{j,i}x^{i+j} \pmod{n}$$

$$\equiv e_{0,0} + e_{i,i}x^{2i} + x^j + x^{i+j} \pmod{n}$$

$$\equiv e_{0,0} + e_{i,i}x^{2i} + (1 + x^i)x^j \pmod{n},$$

and therefore $a_0(x) + a_i(x)x^i \equiv e_{0,0} + e_{i,i}x^{2i} \pmod{n}$. Next, we show an important claim.

**Claim 20.** If

$$a_0(x) + a_i(x) = 0,$$

$$a_0(x) + a_i(x)x^i \equiv e_{0,0} + e_{i,i}x^{2i} \pmod{n},$$

then $a_0(x) = a_i(x) = 0$.

**Proof.** The summation of these equations results with

$$a_i(x)(1 + x^i) \equiv e_{0,0} + e_{i,i}x^{2i} \pmod{n}.$$
It holds that $e_{0,0} = e_{i,i}$ by applying $x = 1$ in the last equation. Assume that $e_{0,0} = e_{i,i} = 1$, so we get that

$$a_i(x)(1 + x^i) \equiv 1 + x^{2i}(\text{mod}x^n - 1).$$

Since $1 + x^{2i} = (1 + x^i)^2$, it holds that

$$(1 + x^i)(1 + x^i + a_i(x)) \equiv 0(\text{mod}x^n - 1).$$

Denote by $p(x)$ the polynomial $p(x) = 1 + x^i + a_i(x)$, and since $p(1) = 0$, it holds that $1 + x|p(x)$. As stated in [4.12], it holds that $\gcd(x^i + 1, M_n(x)) = 1$, and since

$$(1 + x^i)p(x) = (x^n - 1)s(x) = M_n(x)(x + 1)s(x)$$

for some polynomial $s(x)$ over $\mathbb{F}_2$, we deduce that $M_n(x)|p(x)$. Therefore we get that $x^n - 1|p(x)$, however $p(x) \in R_n$, and so we deduce that $p(x) = 0$, that is, $a_i(x) = 1 + x^i$. This results with a contradiction since the coefficient of $x^i$ in $a_i(x)$ is 0. Thus $e_{0,0} = e_{i,i} = 0$ and $a_i(x)(1 + x^i) \equiv 0(\text{mod}x^n - 1).$

Notice that $a_i(x) \in R_n$ and by Claim 20 it also holds $a_i(1) = 0$. Since $\gcd(x^i + 1, M_n(x)) = 1$, we derive that $x^n - 1|a_i(x)$ and since $a_i(x) \in R_n$, we immediately get that $a_i(x) = 0$. Finally, since $a_0(x) + a_i(x) = 0$, we also get that $a_0(x) = 0$, and that completes the proof.

Using Claim 20 we get a contradiction since $e_{j,i} = e_{j,0} = 1$. Therefore, it holds that $a_j(x) = 0$ and since $C_3$ is a sub code of $C_2$, we again get that $a_0(x) = a_i(x) = 0$, and that concludes the proof.

Next, it is proved that the redundancy of the code $C_3$ is exactly $3n - 2$. Every linear code over undirected graphs $U[n,k]$ can be represented by a parity-check matrix of dimension $r \times \left(\begin{array}{c} n+1 \\ 2 \end{array}\right)$ over $\mathbb{F}_q$, when $r \geq \left(\begin{array}{c} n+1 \\ 2 \end{array}\right) - k$. Let $H$ be the parity check matrix of the code $C_3$ of dimension $3n \times \left(\begin{array}{c} n+1 \\ 2 \end{array}\right)$, which is constructed as follows. The first $n$ rows of $H$, denoted by $s_h, h \in [n]$, are formed by the neighborhood constraints $S_h, h \in [n]$. The next set of $n$ rows, denoted by $d_m, m \in [n]$, are formed by the diagonal constraints $D_m, m \in [n]$. Lastly, the last set of $n$ rows, denoted by $t_s, s \in [n]$, are formed by the constraints $T_s, s \in [n]$. For a vector $u \in \mathbb{F}_2^n$ denote by $w(u)$ its Hamming weight, i.e., the number of non-zero entries in $u$.

**Theorem 29.** The redundancy of the code $C_3$ is $3n - 2$.

**Proof.** By the Singleton bound in [4.1], the redundancy of $C_3$ is at least $3n - 3$ and therefore $\text{dim(\ker H^T)} \leq 3$. Our goal is to show that $\text{dim(\ker H^T)} = 2$. Let $0, 1$ be a length-$n$ vector of zeros, ones, respectively. Denote by $u_0, u_1$ the following vector of length $3n$.

$$u_0 = (1, 0, 0), u_1 = (0, 0, 1),$$

94
respectively. By [4.5] and [4.9], the sum of the first, last \( n \) rows of \( H \) is zero, and thus it holds that the vector \( u_0, u_1 \) is in ker \( H^T \), respectively. Clearly \( u_0 \) and \( u_1 \) are linearly independent and therefore \( \text{dim}(\ker H^T) \geq 2 \). Furthermore, every self-loop \( e_{m,m}, m \in [n] \) appears only in the constraint \( D_m, m \in [n] \), and thus only vectors of the form \( (x, 0, y) \) can be in ker \( H^T \), where \( x, y \) is a binary vector of length \( n \), respectively.

Now, assume in the contrary that \( \text{dim}(\ker H^T) = 3 \). Therefore, we can deduce that there is another vector \( u_2 = (x, 0, y) \in \ker H^T \) which is not in span\{\( u_0, u_1 \)\}. Since the redundancy of the double-node-erasure-correcting code \( C_2 \) is \( 2n - 1 \), it holds that every \( n - 1 \) rows of the first \( n \) rows of \( H \) are linearly independent. Hence, \( y \notin \{0, 1\} \) and it can be similarly proved that \( x \notin \{0, 1\} \). Another observation is that we can choose a vector \( x \) such that \( w(x) \leq (n-1)/2 \), since otherwise we can choose a vector \( x' \) such that \( (x', 0, y) = (x, 0, y) + u_0 \) which is also in ker \( H^T \) and \( w(x') \leq (n-1)/2 \). The same property holds for the vector \( y \) and thus it is possible to choose vectors \( x_0 \) and \( y_0 \) such that \( w(x_0), w(y_0) \leq (n-1)/2 \) and \( u_2 = (x_0, 0, y_0) \in \ker H^T \). Let \( x_1, y_1 \) be a vector which results from a single right cyclic shift of the vector \( x_0, y_0 \). By symmetry of the construction we can relabel the nodes of every graph in the code, and deduce that the vector \( u_3 = (x_1, 0, y_1) \) is also in ker \( H^T \). Since \( w(x_0) = w(x_1) \leq (n-1)/2 \), there are indices \( i, j \) such that \( (x_0)_i = 1, (x_1)_i = 0 \) and \( (x_0)_j = 0, (x_1)_j = 1 \) and therefore, the vector \( u_3 \) is not in span\{\( u_0, u_1, u_2 \)\}. Thus \( \text{dim}(\ker H^T) \geq 4 \) and we get a contradiction.

### 4.6 Decoding of the Triple-Node-Erasure-Correcting Codes

In Section [4.5] we proved that the code \( C_3 \) can correct the failure of any three nodes in the graph. Note that whenever three nodes fail, the number of unknown variables is \( 3n - 3 \), and so a naive decoding solution for the code \( C_3 \) is to solve the linear equation system of \( 3n - 2 \) constraints with the \( 3n - 3 \) variables. In this section we show how to efficiently solve this linear equation system for \( C_3 \) with time complexity \( \Theta(n^2) \). Clearly, this time complexity is optimal since the complexity of the input size of the graph is \( \Theta(n^2) \).

Assume that \( G \) is a graph in the code \( C_3 \) and that the failed nodes are \( v_0, v_i, \) and \( v_j \). Let \( v \) be a binary vector of length \( 3n - 2 \) denoted by

\[
v = (e_{0,0}, e_{0,1}, \ldots, e_{0,n-1}, e_{i,1}, \ldots, e_{i,n-1},
   e_{j,1}, \ldots, e_{j,i-1}, e_{j,i+1}, \ldots, e_{j,n-1}).
\]

Using the \( 3n - 2 \) constraints in the code \( C_3 \), it is possible to form the linear equation system as

\[
H \cdot v = s.
\]
Here, $H$ is a binary $(3n - 2) \times (3n - 3)$ matrix that its columns are indexed by the edges in $v$ and its rows indexed by the constraints $S_0, S_1, \ldots, S_{n-2}, D_0, D_1, \ldots, D_{n-1}, T_0, T_1, \ldots, T_{n-2}$, and $s$ is a syndrome vector of length $3n - 2$, indexed by the same order of the $3n - 2$ constraints, that is calculated from the surviving edges. In this case, the binary matrix has $H$ has $O(n)$ non zero entries (three rows with $n - 1$ one's and all other rows with at most 6 one’s) and furthermore, it has a unique solution since we proved the minimum distance of the code is four. Hence, according to Wiedemann [19], the vector $v$ can be found in time complexity $O(n^2)$.

4.7 Conclusion

In this paper we continued our research on codes over graphs from [23, 24]. We presented an optimal binary construction for codes correcting a failure of two nodes together with a decoding procedure that its complexity is optimal. We then extended this construction for triple-node-erasure-correcting codes which are at most a single bit away from optimality with respect to the Singleton bound.

Acknowledgments

The authors would like to thank Gil Kupfer for his contribution to the result of Corollary 9 in Appendix B. They also thank the two anonymous reviewers and the associate editor Sudhir Ghorpade for their valuable comments and suggestions, which have contributed for the clarity of the paper and its presentation.

Appendix A

Remember that $i$ and $j$ are indices such that $i, j \in [n] \setminus \{0\}$.

Lemma 27. The following equations hold

(a) $a_j(x)(1 + x^i) + a_j^2(x) \equiv e_{j,j}(1 + x^j)(x^i + x^j)$ (mod $x^n - 1$).

(b) $a_i(x)(1 + x^j) + a_i^2(x) \equiv e_{i,i}(1 + x^i)(x^i + x^j)$ (mod $x^n - 1$).

(c) $a_0(x)(x^i + x^j) + a_0^2(x) \equiv e_{0,0}(1 + x^i)(1 + x^j)$ (mod $x^n - 1$).

Proof. We only prove equation 3 while the other two hold by relabeling of the construction. First we prove two useful properties on polynomials.
Claim 21. The following equation holds

\[(x^{2i} + x^{2j})(1 + x^i) + (1 + x^{2j})(1 + x^i)x^{2i} + (x^{2i} + x^{2j})^2 + (1 + x^{2j})^2 x^i = (1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j}) + (1 + x^i)^3 x^i.\]

Proof. This equation can be rewritten by

\[(x^{2i} + x^{2j})(1 + x^i) + (1 + x^{2j})(1 + x^i)x^{2i} + (1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j}) + (1 + x^i)^2 x^i = (x^{2i} + x^{2j})^2 + (1 + x^{2j})^2 x^i,\]

or,

\[(1 + x^i)(x^{2i} + x^{2j}) + (1 + x^{2j})x^{2i} + (1 + x^j)(x^{2i} + x^{2j}) + (1 + x^i)^2 x^i = (x^{2i} + x^{2j})^2 + (1 + x^{2j})^2 x^i.\]

Moreover, it can be rewritten by

\[(1 + x^i)x^{2i} + x^{2j} + x^{2i} + x^{2j+2i} + (1 + x^{2j})(x^{2i} + x^{2j}) + x^i + x^{3i} = (x^{2i} + x^{2j})^2 + (1 + x^{2j})^2 x^i,\]

or

\[(1 + x^i)x^{2j} + x^{2j+2i} + x^{2i} + x^{2j} + x^{3i+2j} + x^{4j} + x^i + x^{3i} = x^{4i} + x^{4j} + x^i + x^{4j+i}.\]

We finally rewrite it by

\[(1 + x^i)(x^{2i} + x^{4j} + x^i + x^{3i}) = x^{4i} + x^{4j} + x^i + x^{4j+i},\]

which holds since \((1 + x^i)(x^{2i} + x^{4j} + x^i + x^{3i}) = x^{2i} + x^{4j} + x^i + x^{3i} + x^{3i} + x^{4j+i} + x^{2i} + x^{4i} = x^{4i} + x^{4j} + x^i + x^{4j+i}\).

Let \(e_{0,0}, e_{i,i}, e_{j,j}\) be the label on the self-loop of node \(v_0, v_i, v_j\), respectively, and let \(e_{i,0}, e_{j,0}, e_{j,i}\) be the label on the edge \((v_i, v_0), (v_j, v_0), (v_j, v_i)\), respectively.

We define the following three polynomials

\[p(x) = e_{i,i}(1 + x^{2i}) + e_{j,j}(1 + x^{2j}) + e_{j,i}(1 + x^i)(1 + x^j),\]

\[q(x) = e_{0,0}(1 + x^{2i}) + e_{j,j}(x^{2i} + x^{2j}) + e_{j,0}(x^i + x^j)(1 + x^i),\]

\[s(x) = e_{i,0}(x^i + x^{2i}) + e_{j,0}(x^j + x^{2j}) + e_{j,i}(x^{2i+j} + x^{i+2j}).\]
Claim 22. The following equation holds
\[
q(x)(1 + x^i) + p(x)(1 + x^i)x^{2i} + q^2(x) + p^2(x)x^i + s(x)(1 + x^i)^2 = e_{j, j}(1 + x^j)(1 + x^{2j})(x^{2i} + x^{2j}).
\]

Proof. By the definition,
\[
q(x)(1 + x^i) + p(x)(1 + x^i)x^{2i} + q^2(x) + p^2(x)x^i + s(x)(1 + x^i)^2 =
\]
\[
[e_{0, 0}(1 + x^{2i}) + e_{j, j}(x^{2i} + x^{2j}) + e_{j, 0}(x^i + x^j)(1 + x^i)](1 + x^i)
\]
\[
+ [e_{i, i}(1 + x^{2i}) + e_{j, j}(1 + x^{2j}) + e_{j, i}(1 + x^i)(1 + x^j)]
\]
\[
\cdot (1 + x^i)x^{2i}
\]
\[
+ [e_{0, 0}(1 + x^{2i}) + e_{j, j}(x^{2i} + x^{2j}) + e_{j, 0}(x^i + x^j)(1 + x^i)]^2
\]
\[
+ [e_{i, i}(1 + x^{2i}) + e_{j, j}(1 + x^{2j}) + e_{j, i}(1 + x^i)(1 + x^j)]^2x^i
\]
\[
+ [e_{i, 0}(x^i + x^{2i}) + e_{j, 0}(x^j + x^{2j}) + e_{j, i}(x^{2i+j} + x^{i+2j})]
\]
\[
\cdot (1 + x^i)^2
\]
\[
= e_{0, 0}(1 + x^i)^3x^i + e_{i, i}[(1 + x^i)^3x^{2i} + (1 + x^i)4x^i]
\]
\[
+ e_{i, 0}(x^i + x^{2i})(1 + x^i)^2
\]
\[
+ e_{j, 0}(1 + x^i)^2[(x^i + x^j) + (x^i + x^j)^2 + (x^j + x^{2j})]
\]
\[
+ e_{j, i}(1 + x^i)^2[(1 + x^i)x^{2i} + (1 + x^j)^2x^i + (x^{2i+j} + x^{i+2j})]
\]
\[
+ e_{j, j}[(x^{2i} + x^{2j})(1 + x^i) + (1 + x^{2j})(1 + x^i)x^{2i} + (x^{2i} + x^{2j})^2 + (1 + x^{2j})^2x^i]
\]
\[
\overset{(a)}{=} e_{0, 0}(1 + x^i)^3x^i + e_{i, i}(1 + x^i)^3x^{2i} + e_{i, 0}(1 + x^i)^3x^i
\]
\[
+ e_{j, 0}(1 + x^i)^2(x^i + x^{2i})
\]
\[
+ e_{j, i}(1 + x^i)^2[x^{2i} + x^{2i+j} + x^i + x^{i+2j} + x^{2i+j} + x^{i+2j}]
\]
\[
+ e_{j, j}[(1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j}) + (1 + x^i)^3x^i]
\]
\[
= e_{0, 0}(1 + x^i)^3x^i + e_{i, i}(1 + x^i)^3x^{2i} + e_{i, 0}(1 + x^i)^3x^i
\]
\[
+ e_{j, 0}(1 + x^i)^3x^i + e_{j, i}(1 + x^i)^3x^i
\]
\[
+ e_{j, j}[(1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j}) + (1 + x^i)^3x^i]
\]
\[
\overset{(b)}{=} e_{j, j}(1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j})
\]
where Step (a) holds since by Claim 21
\[
e_{j, j}[(x^{2i} + x^{2j})(1 + x^i) + (1 + x^{2j})(1 + x^i)x^{2i} + (x^{2i} + x^{2j})^2 + (1 + x^{2j})^2x^i]
\]
\[
= e_{j, j}[(1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j}) + (1 + x^i)^3x^i],
\]
and Step (b) holds since by equation (1.16) \(e_{0, 0} + e_{i, i} + e_{j, j} + e_{i, 0} + e_{j, 0} + e_{j, i} = 0\).
Summing the equation of Lemma 24.1 with the equation of Lemma 24.3 we get

\[ a_i(x)(1 + x^i) + a_j(x)(1 + x^j) \equiv e_{0,0} + e_{i,i}x^{2i} + e_{j,j}x^{2j} + e_{i,0} + e_{i,j}x^i + e_{j,0} + e_{j,i}x^j + e_{i,i}(1 + x^i)(1 + x^i) \]

and since \( e_{0,0} = e_{i,i} + e_{j,j} = e_{i,0} + e_{j,0} + e_{j,i} \) we rewrite it as

\[ a_i(x)(1 + x^i) + a_j(x)(1 + x^j) \equiv e_{i,i}(1 + x^{2i}) + e_{j,j}(1 + x^{2j}) + e_{j,0}(x^i + x^j + x^{i+j}) + e_{j,i}x^{2i}(\text{mod}x^n - 1), \]

Multiplying the equation of Lemma 24.1 by \( x^i \) and adding it to the equation of Lemma 24.3 we get

\[ a_0(x)(1 + x^i) + a_j(x)(x^i + x^j) \equiv e_{0,0} + e_{i,0}x^{2i} + e_{j,j}x^{2j} + e_{i,0}x^{2i} + e_{j,0}(x^i + x^j + x^{i+j}) + e_{j,i}x^{2i}(\text{mod}x^n - 1), \]

and since \( e_{i,0} = e_{0,0} + e_{j,j} = e_{i,0} + e_{j,0} + e_{j,i} \) we rewrite it as

\[ a_0(x)(1 + x^i) + a_j(x)(x^i + x^j) \equiv e_{0,0}(1 + x^{2i}) + e_{i,0}(x^i + x^j)(1 + x^i) \]

Next, we multiply the equation of Lemma 24.3 by \((x^i + 1)^2\). In the left-hand side of this equation we set the value of \( a_0(x)(1 + x^i) \) from equation (22) and the value of \( a_i(x)(1 + x^i) \) from equation (21) to get that

\[ a_0(x)(1 + x^i)^2 + a_i(x)(1 + x^i)^2x^{2i} + a_j(x)(1 + x^i)^2x^{2j} + a_0^2(x)(1 + x^i)^2 + a_i^2(x)(1 + x^i)^2x^i + a_j^2(x)(1 + x^i)^2x^j \]

\[ \equiv [a_j(x)(x^i + x^j) + q(x)](1 + x^i) + [a_j(x)(1 + x^j) + p(x)](1 + x^i)x^{2i} + [a_j(x)(x^i + x^j) + q(x)]^2 + [a_j(x)(1 + x^j) + p(x)]^2x^i + a_j(x)(1 + x^i)^2x^{2i} + a_j^2(x)(1 + x^i)^2x^j(\text{mod}x^n - 1). \]

The right-hand side of this equation is

\[ s(x)(1 + x^i)^2(\text{mod}x^n - 1). \]

Now, we proceed with the calculations, while having on the left-hand side only
the values that depend on $a_j(x)$, so we receive that,

$$a_j(x)[(x^i + x^j)(1 + x^i) + (1 + x^j)(1 + x^i)x^{2i} + (1 + x^i)^2x^{2j}]$$

$$+ a_j^3(x)[(x^i + x^j)^2 + (1 + x^j)^2x^i + (1 + x^i)^2x^j]$$

$$\equiv a_j(x)(1 + x^j)(x^i + x^j)^2 + x^{2i} + x^{2j} + x^{2i+j} + x^{i+2j})$$

$$+ a_j^2(x)(x^i + x^j) + x^{2j} + x^{2i+j} + x^{i+2j})(\text{mod} x^n - 1)$$

$$\equiv a_j(x)(1 + x^i)^2(1 + x^j)(x^i + x^j)$$

$$+ a_j^2(x)(1 + x^i)(1 + x^j)(x^i + x^j)(\text{mod} x^n - 1).$$

The right-hand side of the last equation is rewritten to be

$$q(x)(1 + x^i) + p(x)(1 + x^i)x^{2i} + q^2(x) + p^2(x)x^i$$

$$+ s(x)(1 + x^i)^2(\text{mod} x^n - 1),$$

which is equal to $e_{j,j}(1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j})$ by Claim 22. Combining both sides together we deduce that

$$a_j(x)(1 + x^i)^2(1 + x^j)(x^i + x^j)$$

$$+ a_j^2(x)(1 + x^i)(1 + x^j)(x^i + x^j)$$

$$\equiv e_{j,j}(1 + x^i)(1 + x^{2j})(x^{2i} + x^{2j})(\text{mod} x^n - 1),$$

which can be rewritten by,

$$(1 + x^i)(1 + x^j)(x^i + x^j)$$

$$\cdot [a_j(x)(1 + x^i) + a_j^2(x) + e_{j,j}(1 + x^j)(x^i + x^j)] \equiv 0$$

$$\text{mod} x^n - 1).$$

Lastly, denote by $m(x)$ the polynomial

$$m(x) = a_j(x)(1 + x^i) + a_j^2(x) + e_{j,j}(1 + x^j)(x^i + x^j),$$

where it holds that $1 + x|m(x)$ since $m(1) \equiv 0(\text{mod} x^n - 1)$. Notice that the polynomials $1 + x^i$ and $1 + x^j$ are in $\mathcal{R}_n$ and by [42] they are co-prime to $M_n(x)$. Similarly, the polynomial $x^i + x^j$ is also in $\mathcal{R}_n$ and thus is co-prime to $M_n(x)$ as well. Therefore, we deduce that $M_n(x)|m(x)$ and $m(x) \equiv 0(\text{mod} x^n - 1)$, which leads to,

$$a_j(x)(1 + x^i) + a_j^2(x) \equiv e_{j,j}(1 + x^j)(x^i + x^j)(\text{mod} x^n - 1).$$

$\square$
Appendix B

Lemma 28. It holds that $e_{0,0} + e_{i,i} + e_{j,j} = e_{j,0} + e_{j,0} + e_{j,i} = 0$.

Proof. We start with proving several important claims.

Claim 23. If

$$a_j(x)(1 + x^j) + a_j^2(x) \equiv e_{j,j}(x^j + x^{i+j} + x^{2j}) \pmod{x^n - 1},$$

then for all $s \in [n]$

$$e_{j,s} = e_{j,(2s)_n} + e_{j,(2s-i)_n},$$

and for all $1 \leq t \leq n - 1$ it holds that

$$e_{j,s} = \sum_{\ell=0}^{2^t-1} e_{j,(2^t s - \ell i)_n}.$$  \hfill (26)

Proof. First notice that by calculating the coefficient of $x^{(2s)_n}$ of equation (24) for all $s \in [n]$ such that $(2s)_n \notin \{j, (i+j)_n, (2j)_n\}$ it holds that

$$e_{j,(2s)_n} + e_{j,(2s-i)_n} + e_{j,s} = 0.$$

For $(2s)_n = j$, $(2s)_n = (i+j)_n$, $(2s)_n = (2j)_n$, since the coefficient of $x^j, x^{(i+j)_n}, x^{(2j)_n}$ in $a_j(x), a_j(x)x^j, a_j^2(x)$ is zero, respectively, we deduce that also in this case we can write

$$e_{j,(2s)_n} + e_{j,(2s-i)_n} + e_{j,s} = 0,$$

which proves the correctness of equation (25).

Next, we prove the rest of this claim by induction on $t$ where $1 \leq t \leq n - 1$.

Base: for $t = 1$, as we showed above, by calculating the coefficient of $x^{(2s)_n}$ of equation (24) we deduce that for all $s \in [n]$ it holds

$$e_{j,s} = e_{j,(2s)_n} + e_{j,(2s-i)_n}.$$  \hfill (25)

Step: assume that the claim holds for all $\tau$ where $1 \leq \tau \leq t - 1 \leq n - 2$, that is,

$$e_{j,s} = \sum_{\ell=0}^{2^\tau-1} e_{j,(2^\tau s - \ell i)_n}.$$  \hfill (25)

By the correctness of equation (25) and replacing $s$ with $(2^{\tau}s - \ell i)_n$ we deduce that

$$e_{j,(2^\tau s - \ell i)_n} = e_{j,(2(2^\tau s - \ell i))_n} + e_{j,(2(2^\tau s - \ell i) - i)_n},$$

101
and for $\tau = t - 1$ we get that

$$e_{j,s} = \sum_{\ell=0}^{2^t-1} e_{j,(2^t s - \ell i)_n} = \sum_{\ell=0}^{2^t-1} \left( e_{j,(2^t s - 2\ell i)_n} + e_{j,(2^t s - 2\ell i - i)_n} \right) = \sum_{\ell=0}^{2^t-1} e_{j,(2^t s - \ell i)_n}.$$ 

For $a, t, s \in [n]$ denote by $I_{a,t,s}$ the number

$$I_{a,t,s} = \{(\tau, \ell) \mid \langle 2\tau s - \ell a \rangle_n = \langle a2^{-1} \rangle_n, \tau \in [t], \ell \in [2^t]\}.$$

**Corollary 2.** For all $1 \leq t \leq n - 1$ and $s \in [n]$ it holds that

$$e_{j,s} = \sum_{\ell=0}^{2^t-1} e_{j,(2^t s - \ell i)_n} + e_{j,j} I_{i,t,s}.$$  

**Proof.** By adding the monomial $e_{j,j} x^i$ to equation (24) we get the same expression as in Claim 27a on the right-hand side. According to equation (25), by calculating the coefficient of $x^{\langle 2s \rangle_n}$, for all $s \in [n] \setminus \{\langle 2^{-1}i \rangle_n\}$ we will get $e_{j,s} = e_{j,(2s)_n} + e_{j,(2s-i)_n}$ and for $s = \langle 2^{-1}i \rangle_n$ we will get $e_{j,s} = e_{j,(2s)_n} + e_{j,(2s-i)_n} + e_{j,j}$. According to the modification of equation (25) for $s = \langle 2^{-1}i \rangle_n$, we need to similarly adjust equation (26) by counting the number of times the self-loop $e_{j,j}$ should be added to the equation. Hence, by the same arguments of the proof of Claim 23, we deduce that

$$e_{j,s} = \sum_{\ell=0}^{2^t-1} e_{j,(2^t s - \ell i)_n} + e_{j,j} I_{i,t,s},$$

where by definition, $I_{i,t,s}$ is the number of pairs $(\tau, \ell)$ where $\tau \in [t]$ and $\ell \in [2^t]$ such that $\langle 2\tau s - \ell i \rangle_n = \langle i2^{-1} \rangle_n$.

Next we show another important claim.

**Claim 24.** For all $s \in [n]$ it holds

$$e_{j,s} = e_{j,(i-s)_n} + e_{j,j} (1 + I_{i,n-1,s}).$$

**Proof.** By Corollary 2 we know that for all $t, s \in [n],

$$e_{j,s} = \sum_{\ell=0}^{2^t-1} e_{j,(2^t s - \ell i)_n} + e_{j,j} I_{i,t,s}. $$
Proof.

By definition, for two more claims. First, we define for $t$

$$
\sum_{\ell=0}^{n(h-1)-1} e_{j,(s-\ell)t}\sum_{\ell=0}^{n,h-2} e_{j,(s-\ell)t} + e_{j,j} I_i,_{\frac{n-1}{2},s}
$$

Hence, in Step (a) we noticed that $\sum_{\ell=0}^{n(h-1)-1} e_{j,(s-\ell)t} + e_{j,j} I_i,_{\frac{n-1}{2},s}$. Therefore,

$$
e_{j,s} = \sum_{\ell=0}^{n(h-1)-1} e_{j,(s-\ell)t} + e_{j,j} I_i,_{\frac{n-1}{2},s}
$$

Note that the summation $\sum_{\ell=0}^{n(h-1)-1} e_{j,(s-\ell)t}$ expresses the neighborhood of the $j$th node (including its self-loop) $h-1$ times (i.e., an even number of times). Hence, in Step (a) we noticed that $\sum_{\ell=0}^{n(h-1)-1} e_{j,(s-\ell)t} = 0$. Step (b) holds since $\sum_{\ell=0}^{n-1} e_{j,\ell} = e_{j,j}$.

Our next goal is to show that the value of $I_{i,\frac{n-1}{2},i}$ is even. For that we show two more claims. First, we define for $t \in [\frac{n+1}{2}]$ the indicator bit $x_t$ as follows:

$$
x_t = \begin{cases} 
0 & \text{if } \langle 2^{t-1} \rangle_n < \langle 2^{-1} \rangle_n, \\
1 & \text{if } \langle 2^{t-1} \rangle_n \geq \langle 2^{-1} \rangle_n.
\end{cases}
$$

Claim 25. For all $2 \leq t \leq \frac{n-1}{2}$,

$$I_{i,t,i} - I_{i,t-1,i} = 2(I_{i,t-1,i} - I_{i,t-2,i}) - x_{t-1} + x_t.
$$

Proof. By definition, for $t \in [\frac{n+1}{2}]$, $I_{i,t,i}$ is given by

$$I_{i,t,i} = |\{(\tau,\ell)\langle 2^\tau i - \ell t \rangle_n = \langle 2^{-1} \rangle_n, \tau \in [t], \ell \in [2^\tau]\}|
$$

$$= |\{(\tau,\ell)\langle 2^\tau - \ell t \rangle_n = \langle 2^{-1} \rangle_n, \tau \in [t], \ell \in [2^\tau]\}|
$$

$$= |\{(\tau,m)\langle m \rangle_n = \langle 2^{-1} \rangle_n, \tau \in [t], 1 \leq m \leq 2^\tau\}|
$$

103
Therefore, it holds that for all $2 \leq t \leq \frac{n-1}{2}$

$$I_{t,i} - I_{t-1,i} = |\{m \mid \langle m \rangle_n = \langle 2^1 \rangle_n, 1 \leq m \leq 2^{t-1}\}|$$

$$= \left\lfloor \frac{2^{t-1}}{n} \right\rfloor + x_t = \left\lfloor 2 \cdot \left( \frac{2^{t-2}}{n} \right) \right\rfloor + x_t$$

$$= 2 \left\lfloor \frac{2^{t-2}}{n} \right\rfloor + x_{t-1} + x_t$$

\[ \overset{(a)}{=} 2(I_{t-1,i} - I_{t-2,i}) - x_{t-1} + x_t, \]

where in Step (a) we used the property that $I_{t,i} - I_{t-1,i} = \lfloor \frac{2^{t-2}}{n} \rfloor + x_{t-1}$.

Claim 26. For all $t \in \left[ \frac{n+1}{2} \right]$ it holds that $\langle I_{t,i} + x_t \rangle_2 = 0$.

Proof. We will prove this claim by induction on $t \in \left[ \frac{n-1}{2} \right]$.

Base: For $t = 0$, $\langle 2^0 \rangle_n = 1$ which is smaller than $\langle 2^{-1} \rangle_n$ and indeed $I_{0,i} + x_0 = 0$. Similarly, for $t = 1$, $\langle 2^1 \rangle_n = 2$ which is smaller than $\langle 2^{-1} \rangle_n$ for all prime $n \geq 5$ and therefore again $I_{1,i} + x_1 = 0$ is even.

Step: Assume that the claim holds for $t-1$ where $2 \leq t < \frac{n-1}{2}$. By the induction assumption, we have that $\langle I_{t-1,i} + x_{t-1} \rangle_2 = 0$, and by Claim 25 we know that

$$I_{t,i} - I_{t-1,i} = 2(I_{t-1,i} - I_{t-2,i}) - x_{t-1} + x_t,$$

or similarly

$$I_{t,i} = 3I_{t-1,i} - 2I_{t-2,i} - x_{t-1} + x_t,$$

and therefore

$$\langle I_{t,i} + x_t \rangle_2 = \langle 3I_{t-1,i} + 2I_{t-2,i} + x_{t-1} \rangle_2$$

$$= \langle I_{t-1,i} + x_{t-1} \rangle_2 = 0.$$

Corollary 3. The value of $I_{n-1,2}$ is even.

Proof. By Claim 26, it holds that $x_{n-1} = 0$ since $\langle 2^{n-1-1} \rangle_n = \frac{n-1}{2} < \frac{n+1}{2} = \langle 2^{-1} \rangle_n$ and we immediately deduce that $I_{n-1,2}$ is even.

By Claim 24, we know that

$$e_{j,s} = e_{j,(i-s)} + e_{j,j}(1 + I_{n-1,i,s}).$$

Since $I_{n-1,2}$ is even, we get

$$e_{j,i} = e_{j,0} + e_{j,j}.$$
By symmetry of the construction, we also get
\[
\begin{align*}
    e_{j,i} + e_{i,0} + e_{i,i} &= 0, \\
    e_{j,0} + e_{i,0} + e_{0,0} &= 0.
\end{align*}
\]

The summation of the last three equalities results with
\[
e_{j,j} + e_{i,i} + e_{0,0} = 0,
\]
and since
\[
e_{j,j} + e_{i,i} + e_{0,0} + e_{j,i} + e_{i,0} + e_{j,0} = 0,
\]
we deduce that
\[
e_{j,i} + e_{i,0} + e_{j,0} = 0.
\]
Bibliography


Chapter 5

Codes over Trees

Lev Yohananov and Eitan Yaakobi

Abstract

In graph theory, a tree is one of the more popular families of graphs with a wide range of applications in computer science as well as many other related fields. While there are several distance measures over the set of all trees, we consider here the one which defines the so-called tree distance, defined by the minimum number of edit operations, of removing and adding edges, in order to change one tree into another. From a coding theoretic perspective, codes over the tree distance are used for the correction of edge erasures and errors. However, studying this distance measure is important for many other applications that use trees and properties on their locality and the number of neighbor trees. Under this paradigm, the largest size of code over trees with a prescribed minimum tree distance is investigated. Upper bounds on these codes as well as code constructions are presented. A significant part of our study is dedicated to the problem of calculating the size of the ball of trees of a given radius. These balls are not regular and thus we show that while the star tree has asymptotically the smallest size of the ball, the maximum is achieved for the path tree.

5.1 Introduction

In graph theory, a tree is a special case of a connected graph, which comprises of \( n \) labeled nodes and \( n - 1 \) edges. Studying trees and their properties has been beneficial in numerous applications. For example, in signal processing, trees are used for the representation of waveforms. In programming languages, trees are used as structures to describe restrictions in the language. Trees also represent collections of hierarchical text which are used in information retrieval. In cybersecurity applications trees are used to represent fingerprint patterns. One of the biology applications includes the tree-matching algorithm to com-
pare between trees in order to analyze multiple RNA secondary structures [30]. Trees are also used in the subgraph isomorphism problem which, among its very applications, is used for chemical substructure searching [3].

An important feature when studying trees is defining an appropriate distance function. Several distance measures over trees have been proposed in the literature. Among the many examples are the tree edit distance [31], top-down distance [28], alignment distance [12], isolated-subtree distance [32], and bottom-up distance [34]. These distance measures are mostly characterized by adding, removing, and relabeling nodes and edges as well as counting differences between trees with a different number of nodes. One of the more common and widely used distance, which will be referred in this work as the tree distance [21, 10], considers the number of edit edge operations in order to transform one tree to another. Namely, given two labeled trees over \( n \) nodes, the tree distance is defined to be half of the minimum number of edges that are required to be removed and added in order to change one tree to another. This value is also equivalent to the difference between \( n - 1 \) and the number of edges that the two trees share in common. Despite the popularity of this distance function, the knowledge of its characteristics and properties is quite limited. The goal of this paper is to close on these gaps and study trees under the tree distance from a coding theory perspective. To the best of our knowledge, this direction has not been explored rigorously so far.

Motivated by the coding theory approach, in this work we apply the tree distance, which is a metric, to study codes over trees with a prescribed minimum tree distance. This family of codes can be used for the correction of edge erasures. There are several applications in which such codes can be used. For example, in data structures, a tree is a widely used abstract data type that simulates a hierarchical tree structure [6]. Such tree data structures store the information in nodes and use edges as pointers between them. There are numerous examples for such tree data structures including abstract syntax trees (AST), parsing trees and binary search trees (BST) [6, 13]. AST represent the abstract syntactic structure of source code written in a programming language, while each node of the tree denotes a construct occurring in the source code. Parsing trees represent the syntactic structure of a string according to some context-free grammar. BST trees store in each node a value greater than all the values in the node’s left subtree and less than those in its right subtree. These tree data structures can be implemented such that each node stores a list of pointers to other nodes in the tree. Theoretically, such pointers might have wrong addresses, which affects the reliability of the data structure. By adding redundancy edges and nodes, codes over trees may correct the unexpected pointer mismatches. Another family of applications include data structures such as tries and suffix trees [13] in which the information is stored on the edges rather than the nodes. Such data structures can be implemented by a list of \( n - 1 \) edges which is a list of node pairs together with the information on every edge. Again, theoretically, such an edge list may have failures that can indeed be corrected using classical error-correction codes.
However, these codes will not be cardinality optimal since they do not take advantage of the structure of the tree. For the binary case, using classical error-correction codes, we show in the paper the construction of codes over trees of size $\Omega(n^{n-2d})$ where $d \leq n/2$ corresponds to the minimum tree distance of the code. Using codes over trees we show that it is possible to construct codes of cardinality $\Omega(n^2)$, while the minimum tree distance $d$ approaches $\lfloor 3n/4 \rfloor$ and $n$ is a prime number.

Another interesting problem for the tree distance is the study of the size of balls according to the tree distance. This investigation is useful not only for applying the sphere packing bound on codes over trees, but also for other applications. For example, in [7] it was claimed that recent research on nanotechnology discovered that structures of DNA molecules can be constructed into trees or lattices, and that future synthesis techniques may use physical constraints to enforce tree structures on the written base. The authors of [7] introduced the tree trace reconstruction problem, in which the goal is to reconstruct a tree from several of its copies while each copy can have node deletions. In this case, the size of the tree balls may be useful. Another approach deals with graph matching, i.e., the problem of finding a similarity between graphs [9, 16]. Graph matching is an important tool used for example in computer vision and pattern recognition. One of the problems under this setup is to find a model graph, which represents the prototype symbol, as a subgraph in an input graph that represents a diagram, which is also called subgraph isomorphism problem [16]. If the model graph cannot be found exactly in the input graph, then the goal is to find a subgraph that is close to the model graph while the similarity is determined by edit operations on the nodes and edges. This problem is also studied for trees called subtree isomorphism problem [29], and the size of balls of trees may be useful for this problem. Lastly, one of the classical problems in graph theory is finding a minimum spanning tree (MST) for a given graph. While the MST problem is solved in polynomial time [15, 22], it may become NP-hard under some specific constraints. For example, in the degree-constrained MST problem (d-MST) [23, 24, 14, 35], it is required that the degree of every vertex in the MST is not greater than some fixed value $d$. In another example, the goal is to look for an MST in which the length of the simple path between every two vertices is bounded from above by a given value $D \geq 4$ [25]. One of the common approaches for solving such problems uses evolution algorithms (EA). Under this setup, the goal is to find a feasible tree to the problem by iteratively searching for a candidate tree. This iterative procedure is invoked by using mutation operations over the current tree in order to produce a new candidate tree. These mutation operations typically involve the modification of edges in the tree and as such are highly related to the tree distance. Thus, in order to analyze the complexity of such algorithms, it is necessary to study the size of the balls according to the tree distance. In fact, in [10] the size of the radius-one ball was computed for all trees with at most 20 vertices. According to this computer search, it was observed that the smallest size of the ball is achieved when the tree is a star tree (i.e.,
the tree has one node connected to all other nodes), while the largest for a path
tree (i.e., the tree has two leaves and the degree of all other nodes is two). In
this paper, we establish this result for any number of nodes in the tree as well as
for any radius. Furthermore, it is shown that the size of the radius- \( t \) ball ranges
between \( \Omega(n^{2t}) \) (for a star tree) and \( \mathcal{O}(n^{3t}) \) (for a path tree), while the average
size of all balls is \( \Theta(n^{2.5t}) \).

This paper is organized as follows. In Section 5.2, we formally define the
tree distance and codes over trees as well as several more useful definitions and
properties for balls of trees, that will be defined in the sequel. An edge erasure is
the event in which one of the edges in the tree is erased and a forest is received
with two connected components. This is also extended to the erasure of multiple
edges. If \( t \) edges are erased, then a forest with \( t + 1 \) connected components is
received and the number of such forests is \( \binom{n-1}{t} \). In Section 5.3, we summarize
all main results of the paper. In Section 5.4, by using several known results
on the number of forests with a fixed number of connected components we are
able to derive a sphere packing bound for codes over trees. More specifically,
the size of codes over trees of minimum tree distance \( d \) cannot be greater than
\( \mathcal{O}(n^{n-d-1}) \). In Section 5.5, we study balls of trees. The tree ball of trees of a
given tree \( T \) consists of all trees such that their tree distance from \( T \) is at most
some fixed radius \( t \). These balls are not regular. In this section, these balls are
studied for radius one. Balls with a general radius are studied in Section 5.6. In
Section 5.7, the size of star, path tree ball is presented, respectively. Lastly, in
Section 5.8, for a fixed \( d \) we show a construction of codes over trees of size
\( \Omega(n^{n-2d}) \). It is also shown that it is possible to construct codes of cardinality
\( \Omega(n^2) \), while the minimum distance \( d \) approaches \( \lfloor 3n/4 \rfloor \) and \( n \) is a prime number.
Finally, Section 5.9 concludes the paper.

### 5.2 Definitions and Preliminaries

Let \( G = (V_n, E) \) be a graph, where \( V_n = \{v_0, v_1, \ldots, v_{n-1}\} \) is a set of \( n \geq 1 \)
labeled nodes, also called vertices, and \( E \subseteq V_n \times V_n \) is its edge set. In this paper,
we only study undirected trees and forests. By a slight abuse of notation, every
undirected edge in the graph will be denoted by \( \langle v_i, v_j \rangle \) where the order in this
pair does not matter, i.e., the notation \( \langle v_i, v_j \rangle \) is identical to the notation \( \langle v_j, v_i \rangle \).
Thus, there are \( \binom{n}{2} \) possibilities for the edges and the edge set is defined by

\[
E_n = \{ \langle v_i, v_j \rangle \mid i, j \in [n] \},
\]

where \( [n] \triangleq \{0, 1, \ldots, n-1\} \).

A finite undirected tree over \( n \) nodes is a connected undirected graph with
\( n-1 \) edges. The degree of a node \( v_i \) is the number of edges that are incident to the
node, and will be denoted by \( \deg(v_i) \). Each node of degree 1 is called a leaf. The
set of all trees over \( n \) nodes will be denoted by \( \mathcal{T}(n) \). An undirected graph that
consists of only disjoint union of trees is called a forest. The set of all forests over
nodes with exactly $\delta$ trees will be denoted by $F(n, \delta)$. Denote by $F(n, \delta)$ the size of $F(n, \delta)$. We sometimes use the notation $\{C_0, C_1, \ldots, C_{\delta-1}\} = F \in F(n, \delta)$ to explicitly denote a forest with $t$ connected components (or subtrees) of $F$. Note that $F(n, 1) = T(n)$.

By Cayley’s formula \[1\] it holds that $|T(n)| = n^{n-2}$. The proof works by showing a bijection $F : T(n) \to [n]^{n-2}$, where for every tree $T \in T(n)$, the Prüfer sequence of $T$ is denoted by $F(T) = w_T$. An important property is that for each $T = (V_n, E)$, the number of appearances of node $v_i \in V_n$ in $w_T$ is equal to $\text{deg}(v_i) - 1$.

Definition 30. A code over trees $C_T$, denoted by $T-(n, M)$, is a set of $M$ trees over $n$ nodes. Each tree in the code $C_T$ is called a codeword-tree. The redundancy $r$ of the code $C_T$ is defined by $r = (n - 2) \log(n) - \log(M)$.\[1\]

Every codeword-tree corresponds to unique information that is stored, sent, or read, i.e., the information is the structure of the codeword-tree. The storage of information depends mainly on the application that will be used. For example, in binary search trees \[6\] the information values and pointers that represent edges are stored in nodes. In tries or suffix trees \[13\] the symbols or strings are stored on edges. In order to deal with erasures and errors of edges of trees, we initiate the study of codes over trees as will be defined next.

Definition 31. An erasure of $\rho$ edges in a tree $T \in T(n)$ is the event in which $\rho$ of the edges in $T$ are erased and $T$ is separated into a forest of $\rho + 1$ connected components over $n$ nodes. An error of $\psi$ edges in a tree $T \in T(n)$ is the event in which $\psi$ of the edges in $T$ are replaced with other $\psi$ edges such that we receive a new tree $T' \in T(n)$.

The tree distance for trees is next defined.

Definition 32. The tree distance between two trees $T_1 = (V_n, E_1)$ and $T_2 = (V_n, E_2)$ will be denoted by $d_T(T_1, T_2)$ and is defined to be,

$$d_T(T_1, T_2) = n - 1 - |E_1 \cap E_2|.$$ 

It is clear that $d_T(T_1, T_2) = |E_1 \setminus E_2| = |E_2 \setminus E_1|$. Every tree over $n$ nodes can be represented by a binary vector of length $\binom{n}{2}$ called the characteristic vector. Such a vector is indexed by all possible $\binom{n}{2}$ edges that the tree can have and it has ones only in the indices of the tree’s edges. Using this representation, the tree distance between any two trees is one half the Hamming distance between their characteristic vectors. Thus, the tree distance is a metric as was mentioned in \[21\] and is stated in the next lemma.

Lemma 33. The tree distance is a metric.

\[1\]The base of all logarithms in the paper is assumed to be 2.
The tree distance of a code over trees $C_T$ is denoted by $d_T(C_T)$, which is the minimum tree distance between any two distinct trees in $C_T$, that is,

$$d_T(C_T) = \min_{T_1 \neq T_2, T_1, T_2 \in C_T} \{d_T(T_1, T_2)\}.$$ 

**Definition 34.** A code over trees $C_T$ of tree distance $d$, denoted by $T_-(n, M, d)$, has $M$ trees over $n$ nodes and its tree distance is $d_T(C_T) = d$.

Since the tree distance is a metric the following theorem holds straightforwardly.

**Theorem 35.** A $T_-(n, M)$ code over trees $C_T$ is of tree distance at least $d$ if and only if it can correct any $d - 1$ edge erasures and if and only if it can correct any $\lfloor (d - 1)/2 \rfloor$ edge errors.

Next, we define the largest size of a code over trees with a prescribed tree distance.

**Definition 36.** The largest size of a code over trees with tree distance $d$ is denoted by $A_T(n, d)$. The minimum redundancy of a code over trees will be defined by 

$$r(n, d) = (n - 2) \log(n) - \log(A(n, d)).$$

A tree will be called a star tree (or a star in short) if it has a node $v_i, i \in [n]$ such that $\deg(v_i) = n - 1$, and all the other nodes $v_j, j \in [n], j \neq i$ satisfy $\deg(v_j) = 1$. A path graph or a path tree over $n$ nodes is a graph whose nodes can be listed in the order $v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}}$, where $i_0, i_1, \ldots, i_{n-1} \in [n]$, such that its edges are $\langle v_j, v_{j+1} \rangle$ for all $j \in [n - 1]$.

![The star tree.](image)

![The path tree.](image)

**Figure 5.1:** Presentation for a star and a path trees for $n = 5$.

**Definition 37.** The tree ball of a tree of radius $t$ in $T(n)$ centered at $T \in T(n)$ is defined to be

$$B_T(n, t) = \{T' \in T(n) \mid d_T(T', T) \leq t\}.$$ 

The size of the tree ball of trees of $T$, $B_T(n, t)$, is denoted by $V_T(n, t)$.

Note that $V_T(n, t)$ depends on the choice of its center $T$. For example, we will show that if $T$ is a star then $V_T(n, 1) = (n - 1)(n - 2) + 1$ and if $T$ is a path
tree, then 

\[ V_T(n, 1) = (n - 1)(n - 2)(n + 3)/6 + 1. \]

If \( T \) is a star, path tree the size of \( V_T(n, t) \) is denoted by \( V^*(n, t), V^-(n, t) \), respectively. We define the average ball size of radius \( t \) to be the average value of all tree balls of trees of radius \( t \), that is,

\[ V(n, t) = \frac{\sum_{T \in T(n)} V_T(n, t)}{n^{n-2}}. \]

**Definition 38.** The sphere of radius \( t \geq 0 \) centered at \( T \in T(n) \) is defined to be

\[ S_T(n, t) = B_T(n, t) \setminus B_T(n, t - 1), \]

where \( S_T(n, 0) = B_T(n, 0) = \{ T \} \), by definition. The size of the sphere of radius \( t \) is equal to the number of all trees in \( S_T(n, t) \) and is denoted by \( S_T(n, t) \). If \( T \) is a star, path tree then we denote the sphere \( S_T(n, t) \) by \( S^*(n, t), S^-(n, t) \), respectively.

For each \( T = (V_n, E) \in T(n) \) and for each \( E' \subseteq E, |E'| = t \), denote the forest \( F_{T,E'} = (V_n, E \setminus E') \). Note that \( F_{T,E'} \in F(n, t + 1) \).

**Definition 39.** The forest ball of a tree \( T = (V_n, E) \) of radius \( t \) in \( F(n, t + 1) \) is defined to be

\[ P_T(n, t) = \{ F_{T,E'} \in F(n, t + 1) \mid E' \subseteq E, |E'| = t \}. \]

Given a tree \( T = (V_n, E) \) and an edge-set \( E' \in E, |E'| = t \), let \( F_{T,E'} = (V_n, E \setminus E') \in P_T(n, t) \) be the forest which is also denoted by \( F_{T,E'} = \{ C_0, C_1, \ldots, C_t \} \), such that \( |C_0| \leq |C_1| \leq \cdots \leq |C_t| \). The profile vector of \( T \) and \( E' \) is denoted by \( P_T(E') = (|C_0|, |C_1|, \ldots, |C_t|) \) and the multi-set \( P_T(n, t) \) is given by

\[ P_T(n, t) = \{ P_T(E') \mid E' \subseteq E, |E'| = t \}. \] (5.2)

It is can be verified that \( |P_T(n, t)| = |P_T(n, t)| = \binom{n-1}{t} \).

**Definition 40.** The tree ball of a forest (or the forest’s ball in short) of radius \( t \) centered at \( F \in F(n, t + 1) \) is defined to be

\[ B_F(n, t) = \{ T \in T(n) \mid F \in P_T(n, t) \}. \]

The size of the forest’s ball of radius \( t \) is equal to the number of all trees in \( B_F(n, t) \) and is denoted by \( V_F(n, t) \).

Notice that for every two distinct trees \( T_1, T_2 \in B_F(n, t) \) it holds that

\[ d_T(T_1, T_2) \leq t. \]

Note also that we have three different ball definitions, the forest ball of trees of Definition 38 denoted by \( P_T(n, t) \), the tree ball of trees of Definition 37 denoted by \( B_T(n, t) \), the forest’s ball of Definition 40 denoted by \( B_F(n, t) \).

Furthermore, for the convenience of the reader, relevant notation and terminology referred to throughout the paper is summarized in Table 5.1.
Table 5.1: Table of Definitions and Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>The number of nodes</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$T(n)$</td>
<td>The set of all labeled trees over $n$ nodes</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$F(n, \delta)$</td>
<td>The set of all forests with $\delta$ connected components</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$F(n, \delta)$</td>
<td>The size of $F(n, \delta)$</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$d$</td>
<td>The tree distance</td>
<td>Def. 5.2</td>
</tr>
<tr>
<td>$T(n, M, d)$</td>
<td>A code over trees of size $M$</td>
<td>Def. 34</td>
</tr>
<tr>
<td>$A(n, d)$</td>
<td>The largest size of a $T(n, M, d)$ code</td>
<td>Def. 36</td>
</tr>
<tr>
<td>$r(n, d)$</td>
<td>The minimum redundancy of a $T(n, M, d)$ code</td>
<td>Def. 36</td>
</tr>
<tr>
<td>$t$</td>
<td>The radius of a ball</td>
<td>Def. 37</td>
</tr>
<tr>
<td>$B_T(n, t)$</td>
<td>The tree ball of a tree of radius $t$ centered at $T$</td>
<td>Def. 37</td>
</tr>
<tr>
<td>$V_T(n, t)$</td>
<td>The size of $B_T(n, t)$</td>
<td>Def. 37</td>
</tr>
<tr>
<td>$V(n, t)$</td>
<td>The average ball size of radius $t$</td>
<td>Def. 37</td>
</tr>
<tr>
<td>$V^*(n, t)$</td>
<td>The value of $V_T(n, t)$ if $T$ is a star</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$V^-(n, t)$</td>
<td>The value of $V_T(n, t)$ if $T$ is a path tree</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$S_T(n, t)$</td>
<td>The sphere of a tree of radius $t$ centered at $T$</td>
<td>Def. 38</td>
</tr>
<tr>
<td>$S_T(n, t)$</td>
<td>The size of $S_T(n, t)$</td>
<td>Def. 38</td>
</tr>
<tr>
<td>$S^*(n, t)$</td>
<td>The value of $S_T(n, t)$ if $T$ is a star</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$S^-(n, t)$</td>
<td>The value of $S_T(n, t)$ if $T$ is a path tree</td>
<td>Sec. 5.2</td>
</tr>
<tr>
<td>$P_T(n, t)$</td>
<td>The forest ball of a tree of radius $t$ centered at $T$</td>
<td>Def. 39</td>
</tr>
<tr>
<td>$P_T(n, t)$</td>
<td>The set of profiles of $P_T(n, t)$</td>
<td>Def. 39</td>
</tr>
<tr>
<td>$B_F(n, t)$</td>
<td>The tree ball of a forest of radius $t$ centered at $F$</td>
<td>Def. 40</td>
</tr>
<tr>
<td>$V_F(n, t)$</td>
<td>The size of $B_F(n, t)$</td>
<td>Def. 40</td>
</tr>
</tbody>
</table>

5.3 Main Results

This section summarizes the main results in the paper. Theorem 41 states three main upper bounds which will be presented in Section 5.4. The first bound is a sphere packing bound that will be proved in Theorem 46. The second, third bound is an improved upper bound in case that $d = n - 2, d = n - 3$ that will be derived in Theorem 47, 50, respectively.

Theorem 41. 1. For all $n \geq 1$ and fixed $d$,

$$A(n, d) \leq F(n, d) / \binom{n-1}{d-1} = O(n^{n-1-d}).$$

2. For all positive integers $n$, $A(n, n - 2) \leq n$.

3. For all $n \geq 9$, $A(n, n - 3) \leq n^2$.

While in Theorem 41 we obtained upper bounds on $A(n, d)$ using forest balls of trees, in Theorem 42 we show another approach to obtain both lower and upper
bounds on codes over trees using tree balls of trees. For that, in Section 5.5 tree balls of trees of radius one are studied and the main results on these balls are summarized in the next theorem.

**Theorem 42.** 1. For any $T \in \mathcal{T}(n)$, 
\[ V^*(n, 1) \leq V_T(n, 1) \leq V^-(n, 1). \]

2. For all $n \geq 1$, $V(n, 1) \approx 0.5\sqrt{\frac{2}{\pi}}n^{2.5} = \Theta(n^{2.5})$.

3. For all $n \geq 1$, $V^*(n, 1) = \Theta(n^2)$, $V^-(n, 1) = \Theta(n^3)$.

The first, second result of Theorem 42 is proved in Theorem 52, while the third is deduced using Lemma 51. The reader can verify that the hardest part of this theorem is to prove, using inductive and recursive arguments, that $V_T(n, 1) \leq V^-(n, 1)$. In fact this is shown for arbitrary $t$ in Section 5.7. By using the fact that $V^*(n, 1) = \Theta(n^2)$, the upper bound $A(n, 3) = \mathcal{O}(n^{n-4})$ is concluded which is the same upper bound result as the sphere packing bound. Applying the generalized Gilbert-Varshamov bound, while using the fact that $V(n, 1) = \Theta(n^{2.5})$, it is then deduced that $A(n, 2) = \Omega(n^{n-4.5})$. This bound is improved in Section 5.8

In Section 5.6 similar results, summarized in the next theorem, of tree balls of trees with arbitrary radius are shown.

**Theorem 43.** For all $T \in \mathcal{T}(n)$ and fixed $t$, it holds that

1. $V_T(n, t) = \Omega(n^{2t})$, $V_T(n, t) = \mathcal{O}(n^{3t})$.

2. $V(n, t) = \Theta(n^{2.5t})$.

3. $V^*(n, t) = \Theta(n^{2t})$ and $V^-(n, t) = \Theta(n^{3t})$.

The first result is shown in Theorem 58, the second is deduced in Corollary 10 and the third one is shown in Section 5.4 as a result of Theorem 51. These results are derived from recursive formulas that calculate the size of the tree balls of trees of radius $t$.

Again, using the fact that $V^*(n, t) = \Theta(n^{2t})$, it is deduced that for all $d = 2t + 1$, $A(n, d) = \mathcal{O}(n^{n-1-d})$ which matches the upper bound results by the sphere packing bound. Applying the generalized Gilbert-Varshamov lower bound and using the fact that $V(n, t) = \Theta(n^{2.5t})$, it is also derived that for $d = t + 1$, $A(n, d) = \Omega(n^{n-2-2.5(d-1)})$. This bound is also improved in Section 5.8

In Section 5.7 we study the sizes of tree balls of trees of stars and path trees for arbitrary radius. Our main contribution in this section is formulated

---

1 Let $X$ be a finite set with some distance function $d : X \times X \to \mathbb{N}$. Assume that the volume of every ball is $B_r(x) = \{y \in X | d(x, y) \leq r\}$. It was proved in [33] that if $\overline{\Delta}_r = \left(\sum_{x \in X} |B_r(x)| / |X| \right)$, then the generalized Gilbert-Varshamov bound asserts that there exists a code with minimum distance $r + 1$ and of size at least $|X| / \overline{\Delta}_r$. 

117
in recursive formulas for the sizes of tree balls of trees for arbitrary trees. We then show upper and lower bounds on these formulas using the sizes of tree balls of trees of the star and path trees. We present these results in the following theorem.

**Theorem 44.** For all $n$ and fixed $t$ let

$$P = n^{t-1} \binom{n-1}{t} (n-t), \quad Q = n^{t-1} \binom{n+t}{2t+1}.$$ 

The following properties holds:

1. $$\sum_{i=0}^{t} \binom{n-2-t+i}{i} V^*(n, t-i) = P.$$ 

2. $$\sum_{i=0}^{t} \binom{n-2-t+i}{i} V^-(n, t-i) = Q.$$ 

3. For all $T \in T(n)$

$$P \leq \sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) \leq Q.$$ 

The first result is deduced from Theorem 57 and is also shown in Equation (5.17). The second result is due to Theorem 62 and the last result can be found in the proof of Theorem 65. The reader will find out that the challenging part of this theorem is to prove that

$$\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) \leq Q,$$

which is also used in order to prove that $V_T(n, 1) \leq V^-(n, 1)$. This section concludes with conjecturing that for fixed $t$ and $n$ large enough,

$$V^*(n, t) \leq V_T(n, t) \leq V^-(n, t).$$

Lastly, in Section 5.8 we provide several constructions that improve upon the generalized Gilbert-Varshamov lower bounds. The results of these constructions are summarized in the following theorem.

**Theorem 45.** It holds that
1. There exists an $T-(n, \lfloor n/2 \rfloor, n-1)$ code.

2. There exists an $T-(n, n, n-2)$ code.

3. For any positive integer $d \leq n/2$, there exists an $T-(n, M, d)$ code such that $M = \Omega(n^{n-2d})$.

4. For fixed $m$ and prime $n$, there exists an $T-(n, n-1, 2m)\cdot \lfloor n-1 \rfloor \cdot \lfloor 3n/2m \rfloor - 2)$ code.

The result in a) is proved in Theorem 66 using Construction 11, the result in b) is due to Theorem 67 and Construction 12, the result in c) holds according to Corollary 13 and Construction 13, and the result in d) follows from Construction 14 and is proved in Theorem 71. The result in d) assures that it is possible to construct codes of cardinality $\Omega(n^2)$, while the minimum distance $d$ approaches $\lfloor 3n/4 \rfloor$ and $n$ is a prime number. Comparing to Theorem 50 in which it was shown that $A(n, n-3) = O(n^2)$, the result of Theorem 71 shows that $A(n, d) = \Theta(n^2)$, when $d$ approaches $\lfloor 3n/4 \rfloor$ and $n$ is prime. Thus, finding the range of values of $d$ for which $A(n, d) = \Theta(n^2)$ is left for future work.

5.4 Upper Bounds on Codes over Trees

In this section we show upper bounds for codes over trees. Remember that $F(n, \delta)$ is the size of $\mathcal{F}(n, \delta)$, i.e., the number of forests with $n$ nodes and $\delta$ connected components. The value of $F(n, \delta)$ was shown in [18], to be

$$F(n, \delta) = \binom{n}{\delta} n^{n-\delta-1} \sum_{i=0}^{\delta} \left( -\frac{1}{2} \right)^i \binom{\delta}{i} \frac{(\delta + i)(n-\delta)!}{n^i(n-\delta-i)!}$$

or another representation of it in [11],

$$F(n, \delta) = n^{n-\delta} \sum_{i=0}^{\delta} \left( -\frac{1}{2} \right)^i \binom{\delta}{i} \frac{(n-1)(\delta + i)!}{\delta - 1 + i \cdot n^i \delta!}.$$

The next corollary summarizes some of these known results.

**Corollary 4.** The following properties hold for all $n$.

1. $F(n, 1) = n^{n-2},$
2. $F(n, 2) = \frac{1}{2} n^{n-4}(n-1)(n+6),$
3. $F(n, 3) = \frac{1}{8} n^{n-6}(n-1)(n-2)(n^2 + 13n + 60),$
4. $F(n, n-4) = \frac{1}{16} \binom{n}{4} (n^2 + 3n + 10)(n-4)(n+3),$
5. $F(n, n-3) = \frac{1}{2} \binom{n}{4}(n^2 + 3n + 4),$
6. \(F(n, n - 2) = 3\left(\frac{n+1}{4}\right),\)
7. \(F(n, n - 1) = \binom{n}{2},\)
8. \(F(n, n) = 1.\)

### 5.4.1 Sphere-Packing Bound

The following theorem proves the sphere packing bound for codes over trees.

**Theorem 46.** For all \(n \geq 1\) and \(1 \leq d \leq n\), it holds that \(A(n, d) \leq F(n, d)/\binom{n-1}{d-1}\).

**Proof.** Let \(C_T\) be a \(T-(n, M, d)\) code such that \(n \geq 1\) and \(1 \leq d \leq n\). Using Theorem 35 it is deduced that given a codeword-tree \(T_1\), each \(d-1\) of its edge erasures can be corrected. Thus, every forest \(F\) in the forest ball of trees \(\mathcal{P}_{T_1}(n, d - 1)\) cannot appear in any other forest ball of trees \(\mathcal{P}_{T_2}(n, d - 1)\), for all \(T_2 \in C_T \setminus \{T_1\}\). Thus, for every two distinct codeword-trees \(T_1, T_2 \in C_T\) it holds that

\[\mathcal{P}_{T_1}(n, d - 1) \cap \mathcal{P}_{T_2}(n, d - 1) = \emptyset.\]

As already mentioned, for all \(T = (V, E)\) it holds that \(|\mathcal{P}_T(n, d - 1)| = \binom{n-1}{d-1}\). Therefore,

\[M \cdot \binom{n-1}{d-1} = M \cdot |\mathcal{P}_T(n, d - 1)| \leq F(n, d),\]

which leads to the fact that \(A(n, d) \leq \frac{F(n, d)}{\binom{n-1}{d-1}}\).

\[\square\]

It was also proved in [18] that for any fixed \(\delta\),

\[\lim_{n \to \infty} \frac{F(n, \delta)}{n^{n-2}} = \frac{1}{2^{\delta-1}(\delta-1)!},\]

which immediately implies the following corollary.

**Corollary 5.** For all \(n \geq 1\) and fixed \(d\), it holds that

\[A(n, d) \leq F(n, d)/\binom{n-1}{d-1} = \mathcal{O}(n^{n-1-d}),\]

and thus \(r(n, d) = (d - 1) \log(n) + \mathcal{O}(1)\).

Notice that by Corollary 4[17] it holds that

\[A(n, n - 1) \leq \binom{n}{2}/(n - 1) = n/2.\]  \hspace{1cm} (5.3)
In Section 5.8 we will show that

\[ A(n, n - 1) = \lfloor n/2 \rfloor, \]

by showing a construction of a \( T-(n, n - 1) \) code over trees for all \( n \geq 1 \).

Similarly, by Corollary [6],

\[ A(n, n - 2) = 3 \frac{(n + 1)}{4} \left(\frac{n - 1}{n - 3}\right) = \frac{1}{2} \binom{n + 1}{2}, \tag{5.4} \]

however, we will next show how to improve this bound such that \( A(n, n - 2) \leq n \).

In Section 5.8, a construction of \( T-(n, n, n - 1) \) codes over trees will be shown, leading to \( A(n, n - 2) = n \). Finally, by Corollary [5],

\[ A(n, n - 3) \leq \frac{1}{8} n(n^2 + 3n + 4), \tag{5.5} \]

where a better upper bound will be shown in the sequel, which improves this bound to be \( A(n, n - 3) \leq 1.5n^2 \). Finding a construction for this case is left for future work.

Before we show the improved upper bound for \( A(n, n - 3) \), a few more definitions are presented. The girth of a graph is the length of a shortest cycle contained in the graph. If the graph does not contain any cycles (i.e. it is an acyclic graph), its girth is defined to be infinity. For a positive integer \( n \), let \( E_n \) be the set of all \( \binom{n}{2} \) edges as defined in (5.1). A graph \( G = (U \cup V, \mathcal{E}) \) is a bipartite graph with two sets of nodes \( U \) and \( V \) such that \( U \cap V = \emptyset \) and every edge connects a vertex from \( U \) to a vertex from \( V \), i.e., \( \mathcal{E} \subseteq U \times V \). Reiman’s inequality in [20] and [26] states that if \( |V| \leq |U| \), then every bipartite graph \( G = (U \cup V, \mathcal{E}) \) with girth at least 6 satisfies

\[ |\mathcal{E}|^2 - |U| \cdot |\mathcal{E}| - |V| \cdot |U| \cdot (|V| - 1) \leq 0. \tag{5.6} \]

5.4.2 An Improved Upper Bound for \( A(n, n - 2) \)

According to Theorem 46, \( A(n, n - 2) \leq \frac{1}{2} (\binom{n+1}{2}) \) and in the next theorem this bound will be improved to be \( A(n, n - 2) \leq n \).

**Theorem 47.** For all positive integers \( n \), \( A(n, n - 2) \leq n \).

**Proof.** Let \( C_T \) be a \( T-(n, M, n - 2) \) code. Let \( G = (U \cup V, \mathcal{E}) \) be a bipartite graph such that \( V = C_T, U = E_n \) (defined in (5.1)) and \( (T, e) \in \mathcal{E} \) if and only if the tree \( T \in C_T \) has the edge \( e \in E_n \). Clearly, \( |V| = M, |U| = \binom{n}{2} \) and \( |\mathcal{E}| = M(n - 1) \).

Since \( C_T \) is a \( T-(n, M, n - 2) \) code it holds that for all \( T_1 = (V_{n_1}, E_1), T_2 = (V_{n_2}, E_2) \in C_T, |E_1 \cap E_2| \leq 1 \). That is, there are no two codeword-trees in \( C_T \) that share the same two edges. Hence, there does not exist a cycle of length four in \( G \).
If the girth of $G$ is at least 6 (including the case in which $G$ is acyclic by definition of the girth), by (5.4), for all $n \geq 3$, it holds that $|V| = M \leq \frac{1}{2} \left( \binom{n+1}{2} \right) \leq \binom{n}{2} = |U|$, so the inequality stated in (5.6) will be used next. Since $|V| = M, |U| = \binom{n}{2}$ and $|E| = M(n-1)$,

$$M^2(n-1)^2 - \left( \binom{n}{2} \right) M(n-1) - M \left( \binom{n}{2} \right) (M-1) \leq 0,$$

or equivalently

$$M(n-1) - \frac{n}{2} (M-1) \leq \binom{n}{2},$$

which is equivalent to

$$M \left( \frac{n}{2} - 1 \right) \leq \binom{n}{2} - \frac{n}{2},$$

and since

$$\frac{\binom{n}{2} - \frac{n}{2}}{\left( \frac{n}{2} - 1 \right)} = \frac{n(n-1) - \frac{n}{2}}{\left( \frac{n}{2} - 1 \right)} = n \left( \frac{n}{2} - 1 \right) = n,$$

we deduce that $M \leq n$. \qed

As mentioned above, in Section 5.8 we will show that $A(n, n-2) = n$.

### 5.4.3 An Improved Upper Bound for $A(n, n-3)$

We showed in (5.5) that $A(n, n-3) \leq \frac{1}{8} n(n^2 + 3n + 4) = O(n^3)$. In this section this bound will be improved by proving that $A(n, n-3) \leq n^2$.

Denote by $H_n$ the set of forest-sets

$$H_n = \left\{ F \subseteq \mathcal{F}(n, 2) \mid \forall F_1 = (V_1, E_1), F_2 = (V_2, E_2) \in F, |E_1 \cap E_2| \leq 1 \right\}.$$

**Example 12.** For $n = 4$ we partially show an example of the forest-sets in $H_4$. Given a forest-set $F \in H_4$, every two forests $F_1 = (V_1, E_1) \in F, F_2 = (V_2, E_2) \in F$ hold $|E_1 \cap E_2| \leq 1$.

We start with showing the following lemma.

**Lemma 48.** For $n \geq 9$ and for all $F \in H_n$ it holds that $|F| \leq 2n$.

**Proof.** Let $F$ be a forest-set in $H_n$, and let $G = (U \cup V, E)$ be a bipartite graph such that $V = F, U = E_n$ and $(F, e) \in E$ if and only if the forest $F \in F$ has the edge $e \in E_n$. Clearly $|V| = |F|, |U| = \binom{n}{2}$, and $|E| = |F|(n-2)$. Note that $G$ does not have girth 4 since for all $F_1 = (V_1, E_1), F_2 = (V_2, E_2) \in F$ it holds that $|E_1 \cap E_2| \leq 1$.  

122
Assume that the girth of $G$ is at least 6. We consider the following two cases regarding the sizes of the $V$ and $U$. In the first case, where $|V| \leq |U|$ we receive the bound stated in the lemma and we will show that the latter case cannot hold.

**Case 1:** Assume that $|V| = |F| \leq \binom{n}{2} = |U|$. By (5.6)

$$|F|^2(n - 2)^2 - \binom{n}{2} |F|(n - 2) - |F| \binom{n}{2} (|F| - 1) \leq 0,$$

or equivalently

$$|F|(n - 2)^2 - \binom{n}{2} (|F| - 1) \leq \binom{n}{2} (n - 2),$$

which is equivalent to

$$|F|\left((n - 2)^2 - \binom{n}{2}\right) \leq \binom{n}{2} (n - 2) - \binom{n}{2}.$$ 

Next it is deduced that

$$|F|\left((n - 2)^2 - \binom{n}{2}\right) \leq \binom{n}{2} (n - 3),$$

which is equivalent to

$$|F| \leq \frac{n^3 - 4n^2 + 3n}{n^2 - 7n + 8},$$

and therefore $|F| \leq 2n$ for all $n \geq 9$.

**Case 2:** Assume that $|V| = |F| > \binom{n}{2} = |U|$. Again, since the girth is at least six we have that

$$|F|^2(n - 2)^2 - |F|^2(n - 2) - \binom{n}{2} |F|\left(\binom{n}{2} - 1\right) \leq 0,$$

or equivalently

$$|F|(n - 2)^2 - |F|(n - 2) \leq \binom{n}{2} \left(\binom{n}{2} - 1\right),$$
which is equivalent to
\[ |\mathcal{F}|(n - 3) \leq \frac{\binom{n}{2} - 1}{n - 2}. \]

Hence for all \( n \geq 9 \)
\[ |\mathcal{F}| \leq \frac{n(n^2 - 1)}{4(n - 3)} \leq \binom{n}{2}, \]
which results with a contradiction. \( \square \)

Let \( \mathcal{C}_T \) be a \( T-(n, M, n-3) \) code. For all \( e \in E_n \), denote by \( c(\mathcal{C}_T, e) \) the number of codeword-trees of \( \mathcal{C}_T \) having the edge \( e \).

**Lemma 49.** Let \( \mathcal{C}_T \) be a \( T-(n, M, n-3) \) code, where \( n \geq 9 \). Then, for all \( e \in E_n \) it holds that \( c(\mathcal{C}_T, e) \leq 2n \).

**Proof.** For \( e \in E_n \), denote \( k = c(\mathcal{C}_T, e) \) and let \( T_0 = (V_n, E_0), T_1 = (V_n, E_1), \ldots, T_{k-1} = (V_n, E_{k-1}) \in \mathcal{C}_T \) be the \( k \) codeword-trees such that
\[ e \in \bigcap_{i \in [k]} E_i. \] (5.7)
Denote by \( \mathcal{F} \subseteq \mathcal{F}(n, 2) \) the set of \( k \) different forests received by removing the edge \( e \) from \( T_0, T_1, \ldots, T_{k-1} \). Notice that since \( \mathcal{C}_T \) is a \( T-(n, M, n-3) \) code it holds that \( |E_i \cap E_j| \leq 2, i, j \in [k] \) and by (5.7) we deduce that for all distinct \( F_i = (V_n, E_i), F_j = (V_n, E_j) \in \mathcal{F}, |E_i \cap E_j| \leq 1 \). By Lemma 48, for all \( n \geq 9, k = |\mathcal{F}| \leq 2n \) which leads to the fact that \( c(\mathcal{C}_T, e) \leq 2n \). \( \square \)

Lastly, the main result for this section is shown.

**Theorem 50.** For all \( n \geq 9 \), \( A(n, n-3) \leq n^2 \).

**Proof.** Let \( n \geq 9 \) and let \( \mathcal{C}_T \) be a \( T-(n, M, n-3) \) code over trees. Since for all \( e \in E_n \), \( c(\mathcal{C}_T, e) \) is the number of codeword-trees of \( \mathcal{C}_T \) having the edge \( e \), we deduce that \( \sum_{e \in E_n} c(\mathcal{C}_T, e) = M(n - 1) \). By Lemma 49, for all \( e \in E_n \), \( c(\mathcal{C}_T, e) \leq 2n \). Therefore,
\[ M(n - 1) = \sum_{e \in E_n} c(\mathcal{C}_T, e) \leq \binom{n}{2} \cdot 2n = n^2(n - 1), \]
and therefore, \( M \leq n^2 \). \( \square \)

Lastly, we verified that for \( 4 \leq n \leq 8 \), it holds that \( A(n, n-3) \leq 1.5n^2 \).
5.5 Balls of Trees of Radius One

In previous section we introduced and studied the forest ball of a tree in order to derive a sphere packing bound on codes over trees with a prescribed minimum tree distance. In this section we study the size behavior of tree balls of trees. These results will also be used to apply the generalized Gilbert Varshamov bound \[33\] on codes over trees. We start from some definitions.

Our main goal in this section is to study the size of the radius-one tree ball of trees for all trees. This result is proved in the next lemma.

**Lemma 51.** For any $T \in \mathbf{T}(n)$ it holds that

$$V_T(n, 1) = \sum_{(i, n-i) \in P_T(n, 1)} i(n-i) - 1 + 1. \quad (5.8)$$

**Proof.** Let $T = (V_n, E) \in \mathbf{T}(n)$. For any tree $T' = (V_n, E') \in B_T(n, 1) \setminus \{T\}$, if $e \in E \setminus E'$ and $e' \in E' \setminus E$, then $T'$ is generated uniquely by removing an edge $e$ from $E$, yielding two connected components (subtrees) $\{C_0, C_1\} \in P_T(n, 1), |C_0| \leq |C_1|$, and adding the edge $e' \neq e$ between $C_0$ and $C_1$. Thus,

$$|B_T(n, 1) \setminus \{T\}| = \sum_{(|C_0||C_1|) \in P_T(n, 1)} (|C_0||C_1| - 1).$$

By denoting $|C_0| = i$ and $|C_1| = n-i$,

$$V_T(n, 1) = \sum_{(i, n-i) \in P_T(n, 1)} i(n-i) - 1 + 1.$$

\[\square\]

Note that if $T$ is a star, then

$$P_T(n, 1) = \left\{ (1, n-1), \ldots, (1, n-1) \right\}.$$ 

Therefore,

$$V^*(n, 1) = \sum_{(1, n-1) \in P_T(n, 1)} \left( 1 \cdot (n-1) - 1 \right) + 1$$

$$= (n-1)(n-2) + 1.$$ 

If $T$ is a path tree, for odd $n$,

$$P_T(n, 1) = \left\{ (i, n-i), (i, n-i) \mid 1 \leq i \leq \frac{n-1}{2} \right\},$$
and for even \(n\),
\[
P_T(n, 1) = \left\{ (i, n - i), (i, n - i) \mid 1 \leq i \leq \frac{n-2}{2} \right\} \cup \{(n/2, n/2)\}.
\]

In both cases,
\[
V^-(n, 1) = \sum_{i=1}^{n-1} \left( i \cdot (n - i) - 1 \right) + 1
\]
\[
= \sum_{i=1}^{n-1} \left( i \cdot (n - i) \right) - (n - 1) + 1
\]
\[
= \left( \frac{n+1}{3} \right) - (n - 1) + 1
\]
\[
= (n + 1)n(n - 1)/6 - 6(n - 1)/6 + 1
\]
\[
= (n - 1)(n^2 + n - 6)/6 + 1
\]
\[
= (n - 1)(n - 2)(n + 3)/6 + 1,
\]
where \((a)\) and its general case is shown in the proof of Theorem 52.

Our next goal is to show that for any \(T \in \mathcal{T}(n)\) it holds that
\[
V^*(n, 1) \leq V_T(n, 1) \leq V^-(n, 1).
\]

The following claim is easily proved.

**Claim 27.** Given positive integers \(i, n\) such that \(i \in [n]\), it holds that \(n - 1 \leq i(n - i)\).

Next we state that for all \(T \in \mathcal{T}(n)\),
\[
\sum_{(i, n-i) \in P_T(n, 1)} i(n - i) \leq \left( \frac{n+1}{3} \right),
\]
while the proof will be shown in the general case in Lemma 54 where more than one edge is erased.

**Theorem 52.** For any \(T \in \mathcal{T}(n)\) it holds that
\[
V^*(n, 1) \leq V_T(n, 1) \leq V^-(n, 1).
\]

**Proof.** First we prove the lower bound. For all \(T \in \mathcal{T}(n)\)
\[
V_T(n, 1) = \sum_{(i, n-i) \in P_T(n, 1)} \left( i \cdot (n - i) - 1 \right) + 1
\]
\[
\geq \sum_{(i, n-i) \in P_T(n, 1)} \left( 1 \cdot (n - 1) - 1 \right) + 1
\]
\[
= (n - 1)(n - 2) + 1 = V^*(n, 1),
\]
where the inequality holds due to Claim 27. Next, due to (5.9),

\[ V_T(n, 1) = \sum_{(i,n-i) \in P_T(n,1)} (i \cdot (n - i) - 1) + 1 \]

\[ = \sum_{(i,n-i) \in P_T(n,1)} (i \cdot (n - i)) - (n - 1) + 1 \]

\[ \leq \binom{n+1}{3} - (n - 1) + 1 \]

\[ = (n - 1)(n - 2)(n + 3)/6 + 1 = V^-(n, 1), \]

which leads to the fact that \( V_T(n, 1) \leq V^-(n, 1). \)

Our next goal is to show an approximation for the average ball of radius one, that is, the value \( V(n, 1). \) The first step in this calculation is established in the next lemma, where its proof can be found in Appendix A.

**Lemma 53.** For a positive integer \( n \) it holds that

\[ \sum_{T \in T(n)} V_T(n, 1) = \sum_{F \in F(n,2)} (V_F(n, 1))^2 - (n - 2)n^{n-2}. \]

In proof of Lemma 53 we use the equality

\[ \sum_{T \in T(n)} \sum_{F \in F_P(n,1)} 1 = \sum_{F \in F(n,2)} \sum_{T \in B_F(n,1)} 1. \] (5.10)

which holds by changing the order of summation of all distinct couples of trees and forests. One can check that (5.10) is true also for \( t > 1, \) and we will use it in Lemma 59 which is in the next section. Notice also that from this equality it is deduced that

\[ \sum_{F \in F(n,t+1)} V_F(n, t) = \binom{n-1}{t} n^{n-2}. \]

Now, we are ready to show the following theorem.

**Theorem 54.** For all \( n, \)

\[ \sum_{T \in T(n)} V_T(n, 1) = \frac{1}{2} n! \sum_{k=0}^{n-2} \frac{n^k}{k!} - (n - 2)n^{n-2}. \]

**Proof.** It was shown in [18] that

\[ F(n, 2) = \frac{1}{2} \sum_{i=1}^{n-1} \binom{n}{i} i^{i-2} (n - i)^{n-i-2}, \]
where $i$ and $n - i$ represent the sizes of two connected components of each forest in $F(n, 2)$. Furthermore, since for all $\{C_0, C_1\} = F \in F(n, 2)$, if $|C_0| = i$ then $V_F(n, 1) = i(n - i)$, it is deduced that,

$$\sum_{F \in F(n, 2)} (V_F(n, 1))^2 = \frac{1}{2} \sum_{i=1}^{n-1} \left( \binom{n}{i} i^{i-2} (n - i)^{n-i-2} [i(n - i)]^2 \right)$$

$$= \frac{1}{2} \sum_{i=1}^{n-1} \left( \binom{n}{i} i^{i-1} (n - i)^{n-i} \right) \frac{1}{2} n! \sum_{k=0}^{n-2} \frac{n^k}{k!},$$

where (a) holds according to Theorem 5.1 in [2]. Using Lemma 53 it is deduced that

$$\sum_{T \in T(n)} V_T(n, 1) = \frac{1}{2} n! \sum_{k=0}^{n-2} \frac{n^k}{k!} - (n - 2)n^{n-2}.$$

For two functions $f(n)$ and $g(n)$ we say that $f(n) \approx g(n)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$.

As a direct result of Theorem 54 the next corollary follows.

Corollary 6. It holds that,

$$V(n, 1) \approx 0.5 \sqrt{\frac{\pi}{2} n^{2.5}}.$$

Proof. It was shown in [8] that

$$n! \sum_{k=0}^{n-2} \frac{n^k}{k!} \approx \sqrt{\frac{\pi}{2} n^{n+0.5}},$$

and therefore,

$$V(n, 1) = \frac{\sum_{T \in T(n)} V_T(n, 1)}{n^{n-2}} \approx \frac{1}{2} \sqrt{\frac{\pi}{2} n^{n+0.5} - (n-2)}$$

$$= \frac{1}{2} \sqrt{\frac{\pi}{2} n^{2.5}}.$$

To summarize the results of this section, we proved that for every $T \in T(n)$ it holds that $V_T(n, 1) = \Omega(n^2)$, $V_T(n, 1) = O(n^3)$ and the average ball size satisfies $V(n, 1) = \Theta(n^{2.5})$. In order to apply the sphere packing bound for the tree balls of trees of radius one, we can only use the lower bound $V_T(n, 1) = \Omega(n^2)$ and get that

$$A(n, 3) \leq \frac{n^{n-2}}{\alpha n^2} = \frac{1}{\alpha} n^{n-4},$$

128
for some constant $\alpha$. This bound is equivalent in its order to the one achieved in Corollary 5.

While we could not use the average ball size in applying the sphere packing bound, this can be done for the generalized Gilbert-Varshamov lower bound \[33\]. Intuitively, it is done by dividing the number of all trees over $n$ nodes by the average ball size. Namely, according to \[33\], the following lower bound on $A(n, 2)$ holds

$$A(n, 2) = \Omega(n^{-2.5}) = \Omega(n^{-4.5}).$$

The reader can find Construction 13 in Section 5.8 for codes over trees with tree distance $d$ and cardinality $\Omega(n^{-2d})$. In case that $d = 2$ the cardinality is

$$\Omega(n^{-4}) = \Omega(n^{-4}),$$

which improves upon the generalized Gilbert-Varshamov lower bound of this case. In the next section, we show similar results of the ball $B_T(n, t)$ for general radius $t$.

### 5.6 Balls of Trees of Arbitrary Radius

The main goal of this section is to calculate for each $T \in T(n)$ the size of its ball $B_T(n, t)$ and sphere $S_T(n, t)$ for general radius $t$. For that, in Subsection 5.6.1 it is first shown how to calculate the forest’s ball. Using this result, in Subsection 5.6.2 a recursive formula for the tree ball of trees is given and finally in Subsection 5.6.3 we study the average ball size of trees.

#### 5.6.1 The Size of the Forest’s Ball

In this subsection it is shown how to explicitly find the size of the forest’s ball $B_F(n, t)$. By using this result, we will be able to proceed to the next step, which is calculating the size of the tree ball of trees $B_T(n, t)$. Throughout this section we use the notation $\deg_T(v_i)$ for the degree of the node $v_i$ in a tree $T$ in order to emphasize over which tree the degree is referred to. We start with several definitions and claims.

Let $T = (V_t, E) \in T(t)$ be a tree, where $V_t = \{v_0, v_1, \ldots, v_{t-1}\}$, and let $F = \{C_0, C_1, \ldots, C_{t-1}\} \in F(n, t)$ be a forest. Let $P_1(F, T) : F(n, t) \times T(t) \to \mathbb{N}$ be the following mapping. For all $F$ and $T$,

$$P_1(F, T) = \prod_{(v_i, v_j) \in E} |C_i||C_j|.$$

The mapping $P_1$ counts the number of options to complete a forest $F$ with $t$ connected components into a complete tree, according to a specific tree structure $T$ with $t$ nodes, corresponding to the $t$ connected components of $F$. Since every
\(|C_i|\) appears in this multiplication exactly \(\deg_T(v_i)\) times \((v_i \text{ is a node in } T)\), it is deduced that,

\[
P_1(F,T) = \prod_{(v_i,v_j) \in E} |C_i||C_j| = \prod_{C_i \in F} |C_i|^\deg_T(v_i).
\] (5.11)

**Example 13.** Fig. 5.3 demonstrates the mapping \(P_1\). For \(n = 10\) and \(t = 4\), a forest \(F = \{C_0, C_1, C_2, C_3, C_4\} \in F(10, 5)\) over the set of nodes \(\{v_i \mid i \in [10]\}\), and a tree \(T \in T(4)\) over the set of nodes \(\{w_i \mid i \in [5]\}\), are presented. Notice that \(|C_0| = 1, |C_1| = 2, |C_2| = 1, |C_3| = 3, |C_4| = 3,\) and thus, \(P_1(F,T) = |C_0| \cdot |C_1| \cdot |C_0| \cdot |C_2| \cdot |C_0| \cdot |C_2| \cdot |C_3| = 18.\)

![Diagram of a forest and a tree](image)

(a) The forest \(F\).  
(b) The tree \(T\).

Figure 5.3: The mapping \(P_1\).

Let \(F = \{C_0, C_1, \ldots, C_{t-1}\} \in F(n,t)\) be a forest and let \(E_F\) be its edge set. For all \(T = (V_n, E_T) \in V_F(n,t)\) we denote its component edge set \(E_{F,T}\) by

\[E_{F,T} = E_T \setminus E_F.\]

The component edge set is the set of edges that were added to the forest \(F\) in order to receive the tree \(T\). We are ready to show the following claim.

**Claim 28.** For all \(F \in F(n, t + 1)\) it holds that

\[V_F(n,t) = \sum_{T \in T(t+1)} P_1(F,T).\]

*Proof.* Let \(F = \{C_0, C_1, \ldots, C_t\}\) be a forest. Let \(H\) be a mapping \(H : V_F(n,t) \to T(t+1)\) that will be defined as follows. For each \(T \in V_F(n,t)\) with a component edge set \(E_{F,T}\), it holds that \(H(T) = T\) if for all \(e \in E_{F,T}\) such that \(e\) connects between \(C_k\) and \(C_{\ell}\), the edge \(\langle v_k, v_\ell \rangle\) exists in \(T\). Clearly, every \(T \in V_F(n,t)\) is
mapped and $H$ is well defined. Moreover, for any $\mathcal{T} = (V_{t+1}, E) \in \mathcal{T}(t+1)$, $H$ maps exactly

$$
\prod_{(v_i, v_j) \in E} |C_i||C_j|.
$$

trees from $V_F(n, t)$ into $\mathcal{T}$, which is exactly the value of $P_1(F, \mathcal{T})$. Thus,

$$
V_F(n, t) = \sum_{\mathcal{T} \in \mathcal{T}(t+1)} P_1(F, \mathcal{T}).
$$

Next, another mapping $P_2(F, \mathcal{T}) : F(n, t) \times \mathcal{T}(t) \to \mathbb{N}$ is defined. For every forest $F = \{C_0, C_1, \ldots, C_{t-1}\} \in F(n, t)$ and a tree $\mathcal{T} \in \mathcal{T}(t)$ with a prüfer sequence

$$
w_\mathcal{T} = (i_0, i_1, \ldots, i_{t-3}) \in [t]^{t-2},
$$

we let

$$
P_2(F, \mathcal{T}) = |C_{i_0}| \cdot |C_{i_1}| \cdots |C_{i_{t-3}}|.
$$

Using the fact that each number $i$ of node $v_i$ appears in the prüfer sequence $w_\mathcal{T}$ of $\mathcal{T}$ exactly $\deg_\mathcal{T}(v_i) - 1$ times, we deduce that

$$
P_2(F, \mathcal{T}) = \prod_{C_i \in F} |C_i|^{\deg_\mathcal{T}(v_i) - 1}. \tag{5.12}
$$

Let $g_F(x)$ be the generating function of $F$, defined by

$$
g_F(x) = \sum_{i=0}^{t-1} x^{|C_i|}.
$$

This generating function will be used in the proof of the following claim.

**Claim 29.** Let $F$ be a forest in $F(n, t + 1)$. Then,

$$
\sum_{\mathcal{T} \in \mathcal{T}(t+1)} P_2(F, \mathcal{T}) = \left( \sum_{C_i \in F} |C_i| \right)^{t-1} = n^{t-1}.
$$

**Proof.** Let $F \in F(n, t + 1)$ be a forest and let $g_F(x)$ be its generating function. Let $G(x) = (g_F(x))^{t-1}$ and we deduce that

$$
G(x) = (g_F(x))^{t-1} = \left( \sum_{i=0}^{t} x^{|C_i|} \right)^{t-1}
= \sum_{(i_0, i_1, \ldots, i_{t-2}) \in [t+1]^{t-1}} x^{|C_{i_0}| \cdot |C_{i_1}| \cdots |C_{i_{t-2}}|}.
$$

131
Since each monomial of \(G(x)\) is of the from \(x^{C_0} |C_1| \cdots |C_{i-2}|\) for \((i_0, i_1, \ldots, i_{t-2}) \in [t+1]^{t-1}\), it holds that the sum of all the powers of \(x\) in \(G(x)\) is

\[
\sum_{(i_0, i_1, \ldots, i_{t-2}) \in [t+1]^{t-1}} |C_{i_0}| \cdot |C_{i_1}| \cdots |C_{i_{t-2}|},
\]

which is equal to the sum

\[
\left( \sum_{C_i \in F} |C_i| \right)^{t-1}.
\]

Furthermore, each vector \((i_0, i_1, \ldots, i_{t-2}) \in [t+1]^{t-1}\) is a pr"ufer sequence \(w_T\) of some \(T \in T(t+1)\). Thus we deduce that

\[
G(x) = \sum_{T \in T(t+1)} x^{P_2(F,T)},
\]

and the powers sum of \(x\) is exactly

\[
\sum_{T \in T(t+1)} P_2(F,T).
\]

Therefore,

\[
\sum_{T \in T(t+1)} P_2(F,T) = \left( \sum_{C_i \in F} |C_i| \right)^{t-1}.
\]

Lastly, since \(\sum_{C_i \in F} |C_i| = n\), it holds that

\[
\left( \sum_{C_i \in F} |C_i| \right)^{t-1} = n^{t-1},
\]

which concludes the proof. \(\Box\)

According to the last two claims, the next corollary is derived and provides an explicit expression to calculate the forest’s ball size.

**Corollary 7.** For any \(\{C_0, C_1, \ldots, C_t\} = F \in \mathcal{F}(n,t+1)\) it holds that

\[
V_F(n,t) = n^{t-1} \prod_{C_i \in F} |C_i|.
\]

**Proof.** The proof will hold by the following sequence of equations, that will be
explained below,

\[ V_F(n, t) \stackrel{(a)}{=} \sum_{T \in \mathcal{T}(t+1)} P_1(F, T) \]
\[ \quad \stackrel{(b)}{=} \sum_{T \in \mathcal{T}(t+1)} \prod_{C_i \in F} |C_i|^{\deg_T(v_i)} \]
\[ \quad \stackrel{(c)}{=} \prod_{C_i \in F} |C_i| \sum_{T \in \mathcal{T}(t+1)} \prod_{C_i \in F} |C_i|^{\deg_T(v_i) - 1} \]
\[ \quad \stackrel{(d)}{=} \prod_{C_i \in F} |C_i| \sum_{T \in \mathcal{T}(t+1)} P_2(F, T) \]
\[ \quad \stackrel{(e)}{=} \prod_{C_i \in F} |C_i| \left( \sum_{C_i \in F} |C_i| \right)^{t-1} \]
\[ \quad \stackrel{(f)}{=} n^{t-1} \prod_{C_i \in F} |C_i|. \]

Equality (a) holds by Claim 28. Equality (b) holds due to (5.11). Equality (c) is a result of taking the common factor \( \prod_{C_i \in F} |C_i| \) from the summation. Note also that for all \( i \in [t+1], \deg_T(v_i) > 0 \). Equality (d) holds due to (5.12). Equality (e) holds by Claim 29. Equality (f) holds since \( \sum_{C_i \in F} |C_i| + \sum_{C_i \in F} |C_i| + \cdots + |C_t| = n \). □

### 5.6.2 The Size of the Tree Ball of Trees

In this subsection we present a recursive formula for the tree ball of trees \( \mathcal{B}_T(n, t) \) and its sphere \( \mathcal{S}_T(n, t) \), as well as asymptotic bounds on their sizes. First, according to Corollary 7, we immediately get the following corollary.

**Corollary 8.** For all \( T \in \mathcal{T}(n) \) it holds that

\[ \sum_{F \in \mathcal{P}_T(n, t)} V_F(n, t) = n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in \mathcal{P}_T(n, t)} i_0 i_1 \cdots i_t. \]

Next, a recursive connection between the sizes of forest’s balls and spheres (of trees) is shown.

**Lemma 55.** For all \( T \in \mathcal{T}(n) \) it holds that

\[ \sum_{F \in \mathcal{P}_T(n, t)} V_F(n, t) = \sum_{i=0}^{t} \binom{n-1-t+i}{i} S_T(n, t-i). \]  

(5.13)

**Proof.** Let \( T = (V, E) \in \mathcal{T}(n) \). First notice that for all \( 0 \leq i \leq t \),

\[ \bigcup_{i=0}^{t} S_T(n, i) = \bigcup_{F \in \mathcal{P}_T(n, t)} V_F(n, t). \]

Technion - Computer Science Department - Ph.D. Thesis PHD-2022-06 - 2022
Therefore, our main goal in this proof is finding, for a given tree $T_i = (V, E_i) \in S_T(n, i)$, the number of forests in $P_T(n, t)$ in which the tree belongs to their ball of trees. This number equals to the size of the intersection $P_T(n, t) \cap P_{T_i}(n, t)$ since all of these forests belong also to $P_{T_i}(n, t)$. Thus, every forest $F \in P_T(n, t) \cap P_{T_i}(n, t)$ is received in two steps. First, remove from $T_i$ the $t - i$ edges in $E_i \setminus E$. Then, $i$ more edges from $E \cap E_i$ are chosen, where $|E \cap E_i| = n - 1 - (t - i)$. Note that indeed every forest in $P_T(n, t) \cap P_{T_i}(n, t)$ is generated by this procedure. Thus,

$$|P_T(n, t) \cap P_{T_i}(n, t)| = \binom{n - 1 - (t - i)}{i}.$$ 

Therefore, in (5.13) each tree $T_i \in S_T(n, t - i)$ belongs to the forest’s balls of $(n - 1 - (t - i))$ different forests in $P_T(n, t)$. Since it is true for all $0 \leq i \leq t$ we conclude the lemma’s statement.

Combining Corollary 8 and Lemma 55, a recursive formula for the size of a sphere is presented.

**Corollary 9.** For any $T \in T(n)$ it holds that

$$\sum_{i=0}^{t} \binom{n - 1 - t + i}{i} S_T(n, t - i) = n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0 i_1 \cdots i_t.$$ 

Using Corollary 9, a recursive formula for the tree ball of trees is immediately deduced, see Appendix B.

**Theorem 56.** For any $T \in T(n)$ it holds that

$$\sum_{i=0}^{t} \binom{n - 2 - t + i}{i} V_T(n, t - i) = n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0 i_1 \cdots i_t.$$ 

The proof of the following lemma can be found in Appendix C and it is the last step before presenting the main result of this section.

**Lemma 57.** For any positive integer $\alpha$, if

$$\sum_{i=0}^{t} \binom{n - 2 - t + i}{i} V_T(n, t - i) = \Omega(n^{\alpha t}),$$

and $V_T(n, 0) = 1$, then $V_T(n, t) = \Omega(n^{\alpha t})$.

Finally, the main result of this section is shown.

**Theorem 58.** For all $T \in T(n)$ and fixed $t$, it holds that

$V_T(n, t) = \Omega(n^{2t})$, $V_T(n, t) = \mathcal{O}(n^{3t})$. 

134
Proof. First we will prove that \( V_T(n, t) = \Omega(n^{2t}) \). Given positive integers 
\[ i_0, i_1, \ldots, i_{t-1}, i_t, n \]
such that \( i_0 + i_1 + \cdots + i_{t-1} + i_t = n \), it holds that
\[
(n - t) \leq i_0 i_1 \cdots i_t \leq \left( \frac{n}{t+1} \right)^{t+1},
\]
where (a) is well known and (b) holds by using the arithmetic-geometric mean inequality. Thus, for all \( T \in \mathcal{T}(n) \)
\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) \overset{(a)}{=} n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in \mathcal{P}_T(n, t)} i_0 i_1 \cdots i_t \overset{(b)}{\geq} n^{t-1} \binom{n-1}{t} (n-t) = \Omega(n^{2t}),
\]
and
\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) \overset{(a)}{=} n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in \mathcal{P}_T(n, t)} i_0 i_1 \cdots i_t \overset{(b)}{\leq} n^{t-1} \binom{n-1}{t} \left( \frac{n}{t+1} \right)^{t+1} = \mathcal{O}(n^{3t}),
\]
where in both cases (a) holds by Theorem 56 and inequality (b) holds according to (5.14). Therefore, it immediately deduced that \( V_T(n, t) = \mathcal{O}(n^{3t}) \). The result \( V_T(n, t) = \Omega(n^{2t}) \) is deduced according to Lemma 57.

5.6.3 The Average Ball Size

In this section we study the asymptotic behavior of the average ball size (of trees). First, using Theorem 56 and Lemma 55 we deduce that for all \( T \in \mathcal{T}(n) \)
\[
\sum_{F \in \mathcal{P}_T(n, t)} V_F(n, t) = \sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i),
\]
(5.15)
The following recursive relation on the average ball size is presented.
Lemma 59. For all \( n \) and \( t \), it holds that

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V(n, t-i)
\]

\[
= n^{2t-n-1} \frac{1}{(t+1)!} \sum_{i_0, i_1, \ldots, i_t \leq n, i_0 + i_1 + \cdots + i_t = n} \binom{n}{i_0, i_1, \ldots, i_t} i_0^{i_0} i_1^{i_1} \cdots i_t^{i_t}.
\]

Proof. The following holds,

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} \sum_{T \in \mathcal{T}(n)} V_T(n, t-i) \]

\[
= (a) \sum_{T \in \mathcal{T}(n)} \sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i)
\]

\[
= (b) \sum_{T \in \mathcal{T}(n)} \sum_{F \in \mathcal{F}_T(n, t)} V_F(n, t) \]

\[
= (c) \sum_{F \in \mathcal{F}(n, t+1)} \left( V_F(n, t) \right)^2 \]

\[
= (d) \sum_{F \in \mathcal{F}(n, t+1)} (n^{t-1} \prod_{C_i \in F} |C_i|)^2
\]

\[
= (e) \frac{n^{2t-2}}{(t+1)!} \sum_{0<i_0, \ldots, i_t<n, i_0 + i_1 + \cdots + i_t = n} \binom{n}{i_0, \ldots, i_t} i_0^{i_0-2} \cdots i_t^{i_t-2} (i_0 \cdots i_t)^2
\]

\[
= n^{2t-2} \frac{1}{(t+1)!} \sum_{0<i_0, \ldots, i_t<n, i_0 + i_1 + \cdots + i_t = n} \binom{n}{i_0, \ldots, i_t} i_0^{i_0} i_1^{i_1} \cdots i_t^{i_t}.
\]

Equality (a) holds by changing the summation order. Equality (b) holds due to (5.15) and Theorem 56. Equality (c) holds by changing the summation order of trees and forests as it was done in (5.10). Equality (d) holds by Corollary 7.

We deduce equality (e) as follows. It was shown in [18] that

\[
F(n, t+1) = \frac{1}{(t+1)!} \sum_{0<i_0, \ldots, i_t<n, i_0 + i_1 + \cdots + i_t = n} \binom{n}{i_0, \ldots, i_t} i_0^{i_0-2} \cdots i_t^{i_t-2}.
\]

For each \( F \in \mathcal{F}(n, t+1) \) we denote \(|C_j| = i_j, j \in [t+1]\). Thus,

\[
\left( \prod_{C_i \in F} |C_i| \right)^2 = (i_0 \cdots i_t)^2,
\]

which verifies the equality in step (e). After dividing the last expression in the series of equations by \( n^{n-2} \), the proof is concluded. \( \square \)
Next, we seek to show the main result of this section, that is, the asymptotic size of the average ball. For that, we first show the following claim, where its proof is shown in Appendix D.

**Claim 30.** For a positive integer \( n \) and a fixed \( t \) it holds that

\[
\sum_{i=1}^{n-1} \binom{n}{i} i^{(n-i)} \Theta(i^{t/2}) = \Theta(n^{t/2}) \sum_{i=1}^{n-1} \binom{n}{i} i^{(n-i)}. 
\]

The following lemma is now presented.

**Lemma 60.**

\[
\sum_{0<i_0,i_1,...,i_t<n \atop i_0+i_1+...+i_t=n} \binom{n}{i_0,i_1,...,i_t} i_0^{i_0} i_1^{i_1} ... i_t^{i_t} = \Theta(n^{n+t/2}).
\]

**Proof.** Consider the sequence of integers \( 1^1, 2^2, 3^3, \ldots \), that is \( a_n = n^n \), for \( n \geq 1 \). Let \( G(x) \) be its generating function, i.e.

\[
G(x) = \sum_{n=1}^{\infty} a_n \frac{x^n}{n!} = \sum_{n=1}^{\infty} n^n \frac{x^n}{n!}.
\]

Denote by \( F_t(x) \) the function \( F_t(x) = (G(x))^t+1 \). Thus,

\[
F_t(x) = \left( \sum_{i_0=1}^{\infty} \frac{x^{i_0}}{i_0!} \right) \cdots \left( \sum_{i_t=1}^{\infty} \frac{x^{i_t}}{i_t!} \right)
= \sum_{n=1}^{\infty} \left( \sum_{0<i_0,i_1,...,i_t<n \atop i_0+i_1+...+i_t=n} \binom{n}{i_0,i_1,...,i_t} i_0^{i_0} i_1^{i_1} ... i_t^{i_t} \right) \frac{x^n}{n!},
\]

and the coefficient of \( x^n/n! \) in \( F_t(x) \) is exactly

\[
\sum_{0<i_0,i_1,...,i_t<n \atop i_0+i_1+...+i_t=n} \binom{n}{i_0,i_1,...,i_t} i_0^{i_0} i_1^{i_1} ... i_t^{i_t}.
\]

Next, it is shown by induction on \( t \) that the order of the coefficient of \( x^n/n! \) in \( F_t(x) \) is \( \Theta(n^{n+t/2}) \).

**Base:** Clearly the coefficient of \( x^n/n! \) in \( F_0(x) = G(x) \).

**Inductive Step:** Assume that the coefficient of \( x^n/n! \) in \( F_t(x) \) is \( \Theta(n^{n+t/2}) \).
Thus, the coefficient of \(x^n/n!\) in \(F_{t+1}(x)\) is exactly

\[
F_{t+1}(x) = F_t(x)F_0(x)
\]

\[
= \left(\sum_{i=0}^{\infty} \left(\sum_{i_0, i_1, \ldots, i_t \leq n} \binom{n}{i_0, i_1, \ldots, i_t} \frac{x^{i_0} x^{i_1} \cdots x^{i_t}}{i_0! i_1! \cdots i_t!}\right)\right) \cdot \left(\sum_{j=1}^{\infty} j^j \frac{x^j}{j!}\right)
\]

\[
= \left(\sum_{i=1}^{\infty} \binom{n}{i} i^i (n-i)^{n-i} \frac{x^n}{n!}\right) \cdot \left(\sum_{j=1}^{\infty} j^j \frac{x^j}{j!}\right)
\]

where equality (a) holds by the induction assumption, and equality (b) holds by denoting \(i+j=n\). Equality (c) holds by Claim 30 and equality (d) holds due to Corollary 6, where we showed that the coefficient of \(x^n/n!\) in \(F_1(x)\) is \(\Theta(n^{n+0.5})\).

We are now ready to find the asymptotic size of the average ball.

**Corollary 10.** It holds that

\[
V(n, t) = \Theta(n^{2.5t}).
\]

**Proof.** It holds that

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V(n, t-i)
\]

\[
= n^{2t-n} \frac{1}{(t+1)!} \sum_{1 \leq i_0, i_1, \ldots, i_t \leq n, i_0 + i_1 + \cdots + i_t = n} \binom{n}{i_0, i_1, \ldots, i_t} \frac{x^{i_0} x^{i_1} \cdots x^{i_t}}{i_0! i_1! \cdots i_t!}
\]

\[
= \Theta(n^{2t-n}) \Theta(n^{n+t/2}) = \Theta(n^{2.5t}),
\]

where (a) holds by Lemma 59 and (b) holds using Lemma 60. Therefore it is deduced that \(V(n, t) = O(n^{2.5t})\). The result \(V(n, t) = \Omega(n^{2.5t})\) is proved according to Lemma 57.

In summary, we proved that for every \(T \in \mathcal{T}(n)\) and fixed \(t\) it holds that \(V_T(n, t) = \Omega(n^{2t})\), \(V_T(n, t) = O(n^{3t})\) and the average ball size satisfies \(V(n, t) = \Omega(n^{2.5t})\).
\( \Theta(n^{2.5t}) \). The sphere packing bound for smallest tree ball of trees size of radius \( t \) for \( \mathcal{T}-(n, M, d = 2t + 1) \) codes over trees in this case shows that

\[
A(n, d) \leq \frac{n^{n-2}}{\alpha n^{2t}} = \frac{1}{\alpha} n^{n-2-2t} = \frac{1}{\alpha} n^{n-1-d},
\]

for some constant \( \alpha \). Thus, we derive a similar result as in Corollary 5.

By using the generalized Gilbert-Varshamov lower bound for the average ball size \( 33 \) for \( \mathcal{T}-(n, M, d = t + 1) \) codes over trees, we get,

\[
A(n, d) = \Omega(n^{n-2.5(d-1)}) = \Omega(n^{n+0.5-2.5d}).
\]

However, in Section 5.8, based upon Construction 13, we will get that

\[
A(n, d) = \Omega(n^{n-2d}).
\]

In the next section similar results are shown for stars and path trees. While the exact size of the tree balls of trees is found for stars, for path trees we only find its asymptotic behavior and finding its exact expression is left for future work. It is also shown that for a fixed \( t \) the star tree has asymptotically the smallest size of the tree of ball of trees, while the path tree achieves asymptotically the largest size.

### 5.7 The Tree Balls of Trees for Stars and Path Trees

Several more interesting results on the size of the tree balls of trees and more specifically for stars and path trees are shown in this section. First we show an exact formula for \( V^*(n, t) \) and conclude that \( V^*(n, t) = \Theta(n^{2t}) \). Then we simplify the recursive formula in Theorem 56 for path trees and we will show that \( V^-(n, t) = \Theta(n^{3t}) \). Finally, we will show the following explicit upper bound on the recursive formula in Theorem 56, that will not depend on the structure of the tree,

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) \leq n^{t-1} \binom{n+t}{2t+1}.
\]

This result will be shown in Theorem 68.

First, we derive some interesting properties from the recursive formula in Theorem 56, which proved that

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) = n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0 i_1 \cdots i_t.
\]
Notice also that for all $T \in \mathbf{T}(n)$ and $t = n - 1$,
\[
n^{n-2} = n^{(n-1)-1} \sum_{(1,1,\ldots,1) \in P_T(n,n-1)} 1
\]
\[
= \sum_{i=0}^{n-1} \left( n - 2 - (n - 1) + i \right) V_T(n, n - 1 - i)
\]
\[
= \sum_{i=0}^{n-1} \binom{i - 1}{i} V_T(n, n - 1 - i) = V_T(n, n - 1),
\]
where $\binom{-1}{0}$ is defined to be 1, and indeed $V_T(n, n - 1) = n^{n-2}$. Similarly, if $t = n - 2$ then
\[
2(n - 1)n^{n-3} = n^{(n-2)-1} \sum_{(i_0,i_1,\ldots,i_t) \in P_T(n,n-2)} 2
\]
\[
= \sum_{i=0}^{n-2} \binom{i}{i} V_T(n, n - 2 - i) = \sum_{i=0}^{n-2} V_T(n, n - 2 - i),
\]
and thus,
\[
\sum_{i=0}^{n-2} V_T(n, i) = 2(n - 1)n^{n-3}. \tag{5.16}
\]

As for stars, applying Theorem 56, we simply draw the following formula
\[
\sum_{i=0}^{t} \binom{n - 2 - t + i}{i} V^*(n, t - i) = n^{t-1} \binom{n - 1}{t} (n - t). \tag{5.17}
\]

Using this result and the proof of Theorem 56, the following interesting result holds.

**Corollary 11.** For any $T \in \mathbf{T}(n)$ it holds that
\[
\sum_{i=0}^{t} \binom{n - 2 - t + i}{i} \left( V_T(n, t - i) - V^*(n, t - i) \right) \geq 0.
\]

Next an exact formula of the size of the tree ball of trees for stars is presented. The proof of this theorem is shown in Appendix E.

**Theorem 61.** The size of the sphere for a star satisfies
\[
S^*(n,t) = \binom{n - 1}{t} (n - 1)^{t-1}(n - t - 1),
\]
and the size of the tree ball of trees for a star satisfies
\[
V^*(n,t) = \sum_{j=0}^{t} \binom{n - 1}{j} (n - 1)^{j-1}(n - j - 1).
\]
Note that while in Theorem 58 it was shown that for all $T \in \mathcal{T}(n)$ it holds that $V_T(n,t) = \Omega(n^2 t)$, for stars it is deduced that $S^*(n,t) = \Theta(n^{2t})$ and $V^*(n,t) = \Theta(n^{2t})$, which verifies that stars have asymptotically the smallest size of the tree ball of trees.

We turn to study the size of the tree ball of trees for path trees. We first simplify the formula of Theorem 56 in the path tree case.

**Theorem 62.** The size of the tree ball of trees for a path tree satisfies

$$
\sum_{i=0}^{t} \binom{n - 2 - t + i}{i} V^-(n,t-i) = n^{t-1} \binom{n + t}{2t + 1}.
$$

**Proof.** Denote by $A$ the set

$$A = \left\{ (j_0, j_1, \ldots, j_t) \mid \begin{array}{l}
1 \leq j_0 \leq n - t \\
1 \leq j_1 \leq n - (t - 1) - j_0 \\
\vdots \\
1 \leq j_{t-1} \leq n - 1 - \sum_{s=0}^{t-2} j_s \\
j_t = n - \sum_{s=0}^{t-1} j_s
\end{array} \right\}.
$$

Let $T \in \mathcal{T}(n)$ be a path tree. The following equations hold.

$$
\frac{1}{n^{t-1}} \sum_{i=0}^{t} \binom{n - 2 - t + i}{i} V^-(n,t-i) \overset{(a)}{=} \sum_{(i_0, i_1, \ldots, i_t) \in P_T(n,t)} i_0 i_1 \cdots i_t \overset{(b)}{=} \sum_{(j_0, j_1, \ldots, j_t) \in A} j_0 j_1 \cdots j_t \overset{(c)}{=} \binom{n + t}{2t + 1}.
$$

Equality $(a)$ holds due to Theorem 58. As for equality $(b)$, note that after an erasure of $t$ edges of $T$, we get $t + 1$ connected components of $T$ where each of them is a path tree. The value of $j_i$ represents a path subtree as follows. The first path subtree will be of size $j_0$ which can be at least of size 1 and at most of size $n - t$. Similarly, the size $j_1$ of the second path subtree ranges between 1 and $n - (t - 1) - j_0$, i.e. $1 \leq j_1 \leq n - (t - 1) - j_0$. Continuing with this analysis, the size $j_t$ of the last path subtree satisfies $j_t = n - \sum_{s=0}^{t-1} j_s$. Hence, the set of all vectors $(j_0, j_1, \ldots, j_t)$ is exactly the set $A$, which verifies equality $(b)$. Equality $(c)$ holds using combinatorial proof. Consider the problem of counting the number of options to choose $2t + 1$ numbers from the set of numbers $[n+t]$. The right hand side is trivial. As for the left hand side, denote by $(x_1, x_2, \ldots, x_{2t+1})$ a vector such that $x_1 < x_2 < \cdots < x_{2t+1}$ representing an option of chosen $2t + 1$ numbers. We choose these $2t + 1$ in two steps. First we choose the values of $x_2, x_4, \ldots, x_{2t}$.

\[141\]
We translate choosing these numbers to choosing the values of $j_0, j_1, \ldots, j_t$ such that

$$x_2 = j_0, x_4 = j_1 + j_0, \ldots, x_{2t} = \sum_{s=0}^{t-1} j_s = n - j_t.$$ 

In the next step we choose the values of $x_1, x_3, \ldots, x_{2t+1}$. Since $x_1 < x_2$, there are $j_0$ options to pick $x_1$. Similarly since $x_2 < x_3 < x_4$, there are $x_4 - x_2 = j_1$ options to pick $x_3$. Lastly, since $x_{2t} < x_{2t+1} < n$, there are $j_t$ options to pick $x_{2t+1}$. Thus, every option of choosing $j_0, j_1, \ldots, j_t$ counts $j_0j_1 \ldots j_{t-1}j_t$, solutions, and since all options of this problem are counted, the proof is concluded.

Similarly to the case of stars, we showed in Theorem 58 that for all $T \in T(n)$ it holds that $V_T(n, t) = \Theta(n^3)$, and it is also true for path trees as we can see in Theorem 52. According to Lemma 57 we also deduce that $V^-(n, t) = \Theta(n^3)$, that is, a path tree has asymptotically the largest size of the tree ball of trees.

Although a path tree has asymptotically the largest size of the tree ball of trees, it is not necessarily true that for every $n$ and $t$ its size is strictly the largest. We will show such an example at the end of this section.

Our last goal of this section is a stronger upper bound on the size of the tree ball of trees. According to Theorem 58, it was shown that for every tree $T \in T(n)$ it holds that

$$\sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0i_1 \cdots i_t \leq \binom{n - 1}{t} \left(\frac{n}{t+1}\right)^{t+1},$$

while our goal is to improve this upper bound to be

$$\sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0i_1 \cdots i_t \leq \binom{n + t}{2t + 1}. \quad (5.18)$$

While this result does not improve asymptotic upper bound of $O(n^3)$, we believe that this upper bound is interesting and furthermore it verifies the statement of Theorem 55.

First the definition of (5.2) is slightly modified. Let $v_\ell = (v_{i_0}, v_{i_1}, \ldots, v_{i_{t-1}})$ be a vector of $\ell$ not necessary distinct nodes of $T \in T(n)$ where $1 \leq \ell \leq t + 1$. Let $F_{T, E'}(v_\ell) = (V_n, E \setminus E') \in P_T(n, t)$ be a forest which is also denoted by $F_{T, E'}(v_\ell) = \{C_0, C_1, \ldots, C_\ell\}$, such that $v_{i_j} \in C_j, j \in [\ell]$ and $|C_\ell| \leq |C_{\ell+1}| \leq \cdots \leq |C_1|$. In case there is more than one way to order to connected components $C_0, C_1, \ldots, C_\ell$, we choose one of them arbitrarily. For $1 \leq \ell \leq t + 1$ denote the multi-set $P_T(n, t; v_\ell)$

$$P_T(n, t; v_\ell) = \left\{|C_0|, \ldots, |C_\ell| |\{C_0, \ldots, C_\ell\} = F_{T, E'}(v_\ell) \in P_T(n, t)\right\},$$

142
and for $\ell = 0$,

$$P_T(n, t; v_\ell) = P_T(n, t).$$

Intuitively, this multi-set consist of profiles of forests such that all the nodes of the vector $v_\ell$ are in different connected components of these forests. From this definition, in case that not all of the nodes in $v_\ell$ are distinct, then $P_T(n, t; v_\ell) = \emptyset$. Another property is that for all $\ell \in [t + 2]$ it holds that

$$|P_T(n, t; v_\ell)| \leq |P_T(n, t)| = \binom{n-1}{t}.$$

Next, for all $\ell \in [t + 2]$, denote by $f_T(n, t; v_\ell)$ the function

$$f_T(n, t; v_\ell) = \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_\ell)} c_0 c_{t+1} \cdots c_t,$$

where in case that $\ell = t + 1$, the function $f_T$ is defined as

$$f_T(n, t, v_{t+1}) = \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{t+1})} 1 = |P_T(n, t; v_{t+1})| \leq \binom{n-1}{t}.$$

Again, if not all of the nodes in $v_\ell$ are distinct, by definition $P_T(n, t; v_\ell) = \emptyset$ and $f_T(n, t; v_\ell) = 0$.

Since in case that $t = n - 1$, each connected component is of size 1, the following property is immediately deduced,

$$f_T(n, n-1; v_\ell) = \sum_{(c_0, c_1, \ldots, c_{n-1}) \in P_T(n, n-1; v_\ell)} c_0 c_{t+1} \cdots c_{n-1} = 1. \quad (5.19)$$

The main goal in this part is to show that

$$f_T(n, t; v_\ell) \leq \binom{n + t - \ell}{2t + 1 - \ell}, \quad (5.20)$$

where in case that $\ell = 0$, the equality (5.18) is immediately deduced.

Let $T \in \mathcal{T}(n)$ for $n \geq 2$. For two integers $t$ and $\ell$ such that $0 \leq \ell < t + 1 < n$, let $v_\ell = (v_{i_0}, \ldots, v_{i_{t-1}}, v_{i_{t+1}})$ be a vector of $\ell$ nodes in $T$. For $j \in [\ell + 1]$ denote $v_j = (v_{i_0}, \ldots, v_{i_{j-1}})$. For any node $v_x$ in $T$ denote $v_{j+1}(v_x) = (v_{i_0}, \ldots, v_{i_{j-1}}, v_x)$. By a slight abuse of notation, given a vector $(c_0, \ldots, c_t) \in P_T(n, t; v_\ell)$, the connected component $C_j$ is referred to the value $c_j$, or in other words $(c_0, \ldots, c_t) = (|C_0|, \ldots, |C_t|)$. For any node $v_x$ in $T$ denote by $A_T(n, t; v_\ell, v_x)$ the set

$$A_T(n, t; v_\ell, v_x) = \{ (|C_0|, \ldots, |C_t|) \in P_T(n, t; v_\ell) | v_x \in \bigcup_{i \in [\ell]} C_i \}.$$

Let $v_x$ be a leaf connected to a node denoted by $v_y$ in $T \in \mathcal{T}(n)$, and let $T_1 \in \mathcal{T}(n-1)$ be a tree generated by removing $v_x$ from $T$. The definitions introduced above are used in the next claims and lemmas.
Claim 31. The following properties hold

1. It holds that
\[
\sum_{(c_0, \ldots, c_t) \in P_T(n; v_\ell)} c_0 \cdots c_t = \sum_{(c_0, \ldots, c_t) \in A_T(n; v_\ell, v_y)} c_0 \cdots c_t + \sum_{(c_0, \ldots, c_t) \in P_T(n; v_\ell+1(v_y))} c_0 \cdots c_t.
\]

2. If \( v_x \) is not in \( v_\ell \) then,
\[
\sum_{(c_0, \ldots, c_t) \in A_T(n-1; v_\ell, v_y)} c_0 \cdots c_t = \sum_{(c_0', \ldots, c'_t) \in A_T(n; v_\ell, v_\ell)} c'_0 \cdots c'_t.
\]

3. If \( v_x \) is not in \( v_\ell \) then,
\[
\sum_{(c_0, \ldots, c_t) \in P_T(n; v_{\ell+1}(v_x))} c_0 \cdots c_t = \sum_{(c'_0, \ldots, c'_t) \in P_T(n-1; v_{\ell+1}(v_y))} (c'_0 + 1)c'_1 \cdots c'_t + \sum_{(c_0, \ldots, c_{\ell-1}, c_{\ell+1}, \ldots, c_t) \in P_T(n; v_{\ell+1}(v_x))} 1 \cdot c_{\ell+1} \cdots c_t.
\]

4. If \( v_x = v_{i_{\ell-1}} \) then
\[
\sum_{(c_0, \ldots, c_t) \in P_T(n; v_{\ell}(v_x))} c_0 \cdots c_t = \sum_{(c'_0, \ldots, c'_t) \in P_T(n-1; v_{\ell}(v_y))} c'_0 \cdots c'_t + \sum_{(c_0, \ldots, c_{\ell-2}, c_{\ell+1}, \ldots, c_t) \in P_T(n; v_{\ell}(v_x))} c_0 \cdots c_t.
\]

The proof of Claim 31 can be found in Appendix F. Next we show a recursive formula with respect to \( f_T \).

Example 14. In Fig 5.4, we demonstrate the idea of the recursive formula for \( f_T(10, 4; (v_7)) = f_T(9, 3; (v_7)) + f_T(9, 4; (v_7)) + f_T(9, 4; (v_7, v_6)). \)

Lemma 63. If \( v_x \) is not in \( v_\ell \) then,
\[
f_T(n, t; v_\ell) = f_T(n-1, t; v_\ell) + f_T(n-1, t; v_{\ell+1}(v_y)) + f_T(n-1, t-1; v_\ell).
\]

If \( v_x \) is in \( v_\ell \), and without loss of generality \( v_x = v_{i_{\ell-1}} \), then
\[
f_T(n, t; v_\ell) = f_T(n-1, t; v_\ell(v_y)) + f_T(n-1, t-1; v_{\ell-1}).
\]

144
The proof of Lemma 63 can be found in Appendix G. An example that illustrates this recursive formula is now presented.

**Example 15.** For \( n = 10 \), we illustrate in Fig. 5.4(a) a tree \( T \in T(10) \). In this example, \( t = 4 \) and \( \ell = 1 \), \( v_x = v_5, v_y = v_6 \) and \( v_7 = (v_7) \). Let \( T_1 \in T(9) \) be a tree which is derived from \( T \) by removing the node \( v_5 \). After an erasure of 4 edges, the multiplication of the five connected components is counted in \( f(n,t) \). Fig. 5.4(b), (c) and (d) represent the idea of the formula \( f_T(n,t;v_\ell) = f_{T_1}(n-1,t;v_\ell) + f_{T_1}(n-1,t;v_{\ell+1}(v_y)) + f_{T_1}(n-1,t-1;v_\ell) \). The dashed edges in Fig. 5.4(b), (c) and (d) represent the erased edges from \( T \), yielding
a forest with five connected components $C_0, C_1, C_2, C_3,$ and $C_4$. An example of possible erasure including the edge $\langle v_5, v_6 \rangle$ is shown in Fig. 5.4(b). This example emphasizes the case which corresponds to the multiplication $|C_0| \cdot |C_1| \cdot |C_3| \cdot |C_4|$ that is also counted in $f_{T_1}(n-1, t; (v_7))$ since $|C_0| = 1$. Fig. 5.4(c) and (d) similarly emphasize the case in which an erasure of 4 edges does not include the edge $\langle v_5, v_6 \rangle$. While Fig. 5.4(c) emphasizes the multiplication $|C_0| \cdot |C_1| \cdot |C_3| - 1 \cdot |C_4|$, which is counted in $f_{T_1}(n-1, t-1; (v_7))$ (since $v_5$ is not in $T_1$), Fig. 5.4(d) emphasizes the multiplication $|C_0| \cdot |C_1| \cdot |C_3| \cdot |C_4|$, which is counted in $f_{T_1}(n-1, t; (v_7, v_6))$. Hence, $|C_0| \cdot |C_1| \cdot |C_3| \cdot |C_4|$ is also counted in the case that the edge $\langle v_5, v_6 \rangle$ is not erased.

Finally, the upper bound for $f_T(n, t; v_\ell)$ is presented, while the proof is shown in Appendix H.

**Lemma 64.** For any tree $T \in T(n), n \geq 1$ and a vector of $0 \leq \ell \leq t + 1 \leq n$ nodes $v_\ell = (v_{i_0}, v_{i_1}, \ldots, v_{i_{t-1}})$,

$$f_T(n, t; v_\ell) \leq \left( \frac{n + t - \ell}{2t + 1 - \ell} \right).$$

From Lemma 64 it is immediately deduced that for all $T \in T(n)$,

$$\sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0 i_1 \cdots i_t = f_T(n, t) \leq \left( \frac{n + t}{2t + 1} \right). \tag{5.21}$$

Using (5.21) the tighter upper bound for the recursive formula in Theorem 56 is shown in the following theorem.

**Theorem 65.** For any $T \in T(n)$ it holds that

$$\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) \leq n^{t-1} \left( \frac{n + t}{2t + 1} \right).$$

From Theorem 65 and Theorem 62 we immediately deduce the following corollary.

**Corollary 12.** For any $T \in T(n)$ it holds that

$$\sum_{i=0}^{t} \binom{n-2-t+i}{i} \left( V_T(n, t-i) - V^-(n, t-i) \right) \leq 0.$$

Even though by Corollary 11

$$\sum_{i=0}^{t} \binom{n-2-t+i}{i} \left( V_T(n, t-i) - V^*(n, t-i) \right) \geq 0.$$

146
and by Corollary 12
\[ \sum_{i=0}^{t} \binom{n-2-t+i}{i} \left(V_T(n, t-i) - V^{-}(n, t-i)\right) \leq 0, \]
it does not imply that for all \( n \) and \( t \), \( V^{*}(n, t) \leq V_T(n, t) \leq V^{-}(n, t) \). For example, if \( t = n-2 \), \( V^{*}(n, t) = n^{n-2} \) while \( V^{-}(n, t) < n^{n-2} \), since one can check that there are two path trees \( T_1, T_2 \in T(n) \) such that \( d_T(T_1, T_2) = n-1 \). However, we conjecture that for fixed \( t \) and large enough \( n \), it holds that \( V^{*}(n, t) \leq V_T(n, t) \leq V^{-}(n, t) \).

### 5.8 Constructions of Codes over Trees

In this section we show several constructions of codes over trees. The first is the construction of \( T^-(n, \lceil n/2 \rceil, n-1) \) codes, and the second is the construction of \( T^-(n, n, n-2) \) codes. The third and our main result in this section is the construction of \( T^-(n, M, d) \) codes for fixed \( d \) where \( M = \Omega(n^{n-2d}) \). For positive integers \( a \) and \( n \) we will use the notation \( \langle a \rangle_n \) to denote the value of \( a \mod n \).

#### 5.8.1 A Construction of \( T^-(n, \lceil n/2 \rceil, n-1) \) Codes

A path tree \( T = (V_n, E) \) with the edge set
\[ E = \{(v_{ij}, v_{i,j+1}) \mid j \in [n-1], i_j \in [n]\}, \]
will be denoted by \( T = (v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}}) \), i.e., the nodes \( v_{i_0} \) and \( v_{i_{n-1}} \) are leaves and the rest of the nodes have degree 2. Note that the number of path trees over \( n \) nodes is \( n! / 2 \), so every path tree has two representations in this form and we will use either one of them in the sequel.

For \( s \in [[n/2]] \), denote by \( T_s = (V_n, E) \) the path tree
\[ T_s = \begin{cases} \langle v(s)_n, v(s-1)_n, v(s+1)_n, \ldots, v(s+n-1)_n \rangle_n : & \text{if } n \text{ is odd}, \\ \langle v(s)_n, v(s-1)_n, v(s+1)_n, \ldots, v(s-n)_n \rangle_n : & \text{if } n \text{ is even}. \end{cases} \]

**Example 16.** For \( n = 10 \) we show an example of the path tree \( T_0 \). By looking at the lower half of the circle in this figure, i.e. nodes \( v_0, v_9, v_8, v_7, v_6, v_5 \), there is a single edge connecting two vertices on this half circle. The path tree \( T_1 \) is received by rotating anticlockwise the nodes on this circle by one step. Note that all the edges in \( T_0 \) and \( T_1 \) are disjoint and this property holds also for the other path trees \( T_2, T_3, T_4 \).

The construction of a \( T^-(n, \lceil n/2 \rceil, n-1) \) code is given as follows. This construction is motivated by the factorization of the complete graph into mutually disjoint Hamiltonian paths; see [11, 17]. Even though this result is well known, for completeness we present it here along with its proof.
Theorem 66. The code $C_{T_1}$ is a $T$-$(n, \lfloor n/2 \rfloor, n-1)$ code.

Proof. Clearly, since for all distinct $s_1, s_2 \in \lfloor n/2 \rfloor$ it holds that $s_1 \neq s_2 + \lfloor n/2 \rfloor$, it is deduced that $|C_{T_1}| = \lfloor n/2 \rfloor$. Next we prove that this code can correct $\rho = n-2$ edge-erasures, by showing that $d_T(C_{T_1}) > n-2$.

Assume on the contrary that $d_T(C_{T_1}) \leq n-2$. Therefore, there are two distinct numbers $s_1, s_2 \in \lfloor n/2 \rfloor$ such that the trees $T_{s_1} = (V_n, E_1), T_{s_2} = (V_n, E_2) \in C_{T_1}$ hold $|E_1 \cap E_2| \geq 1$. Therefore, there exist two integers $t_1, t_2 \in \lfloor n/2 \rfloor$ such that one of the following cases hold:

1. $\langle v_{s_1+t_1}, v_{s_1-(t_1+1)} \rangle = \langle v_{s_2+t_2}, v_{s_2-(t_2+1)} \rangle$,
2. $\langle v_{s_1+t_1}, v_{s_1-(t_1+1)} \rangle = \langle v_{s_2-t_2}, v_{s_2+t_2} \rangle$,
3. $\langle v_{s_1-t_1}, v_{s_1+t_1} \rangle = \langle v_{s_2-t_2}, v_{s_2+t_2} \rangle$.

We will eliminate all those options as follows.

1. If $\langle s_1+t_1 \rangle = \langle s_2+t_2 \rangle$ and $\langle s_1-(t_1+1) \rangle = \langle s_2-(t_2+1) \rangle$ then by summing those equations we deduce that $\langle 2s_1-1 \rangle = \langle 2s_2-1 \rangle$. Therefore, we deduce that $s_1 = s_2$ which is a contradiction. Similar proof shows that it is impossible to have $\langle s_1+t_1 \rangle = \langle s_2-(t_2+1) \rangle$ and $\langle s_2+t_2 \rangle = \langle s_1-(t_1+1) \rangle$.

2. If $\langle s_1+t_1 \rangle = \langle s_2-t_2 \rangle$ and $\langle s_1-(t_1+1) \rangle = \langle s_2+t_2 \rangle$ then by summing those equations we deduce that $\langle 2s_1-1 \rangle = \langle 2s_2 \rangle$. Since $s_1, s_2 \in \lfloor n/2 \rfloor$, if $s_1 \neq 0$ then $2s_1-1 < n-1$ and $2s_2 < n-1$. Clearly, $\langle 2s_1-1 \rangle$ is odd and $\langle s_2 \rangle$ is even (since both of them smaller than $n$) so it is deduced that they are distinct. If $s_1 = 0$ then $\langle 2s_1-1 \rangle = n-1$ but since $s_2 \in \lfloor n/2 \rfloor$ it holds that $2s_2 < n-1$ and therefore we get again that $\langle 2s_1-1 \rangle \neq \langle 2s_2 \rangle$, which is a contradiction. Similar proof shows that it is impossible to have $\langle s_1+t_1 \rangle = \langle s_2+t_2 \rangle$ and $\langle s_1-(t_1+1) \rangle = \langle s_2-t_2 \rangle$.  

Figure 5.5: Hamiltonian path trees.

Construction 11. For all $n \geq 3$ let $C_{T_1}$ be the following code over trees

$C_{T_1} = \{ T_s = (V_n, E) | s \in \lfloor [n/2] \rfloor \}$. 


3. If \( \langle s_1 - t_1 \rangle_n = \langle s_2 - t_2 \rangle_n \) and \( \langle s_1 + t_1 \rangle_n = \langle s_2 + t_2 \rangle_n \) then by summing those equations we deduce that \( \langle 2s_1 \rangle_n = \langle 2s_2 \rangle_n \). Therefore, we deduce that \( s_1 = s_2 \) which is a contradiction. Similar proof shows that it is impossible to have \( \langle s_1 - t_1 \rangle_n = \langle s_2 + t_2 \rangle_n \) and \( \langle s_1 + t_1 \rangle_n = \langle s_2 - t_2 \rangle_n \).

\[ \]

In this construction the result \( A(n, n - 1) \geq \lfloor n/2 \rfloor \) is shown, and since by (5.3), \( A(n, n - 1) \leq n/2 \) it is deduced that \( A(n, n - 1) = \lfloor n/2 \rfloor \).

5.8.2 A Construction of \( T-(n, n, n - 2) \) Codes

For convenience, a star \( T \) with a node \( v_i \) of degree \( n - 1 \) will be denoted by \( T_{v_i} \). The construction of a \( T-(n, n, n - 2) \) code will be as follows.

Construction 12. For all \( n \geq 4 \) let \( C_{T_2} \) be the following code

\[ C_{T_2} = \{ T_{v_i} = (V_n, E) | i \in [n] \} \]

Clearly, the code \( C_{T_2} \) is a set of all stars over \( n \) nodes. Next we prove that this code is a \( T-(n, n, n - 2) \) code.

Theorem 67. The code \( C_{T_2} \) is a \( T-(n, n, n - 2) \) code.

Proof. Let \( T_{v_i} = (V_n, E), i \in [n], \) be a codeword-tree of \( C_{T_2} \) with a node \( v_i \) of degree \( n - 1 \). Since \( T_{v_i} \) is a star, after the erasure of \( n - 3 \) edges from \( T_{v_i} \), the node \( v_i \) will have degree 2 and all the nodes \( v_j \in T_{v_i}, j \neq i \) will have degree of at most 1. Therefore the node \( v_i \) can be easily recognized and the codeword-tree \( T_{v_i} \) can be corrected.

In this trivial construction we showed that \( A(n, n - 2) \geq n \) and since by Theorem 17 \( A(n, n - 2) \leq n \) it is deduced that \( A(n, n - 2) = n \).

5.8.3 A Construction of \( T-(n, \Omega(n^{n-2d}), d) \) Codes

In this section we show a construction of \( T-(n, \Omega(n^{n-2d}), d) \) codes for any positive integer \( d \leq n/2 \). Note that according to Corollary 3 for fixed \( d \), \( A(n, d) = \Omega(n^{n-1-d}) \) and by Corollary 13 it will be deduced that \( A(n, d) = \Omega(n^{n-2d}) \).

For a vector \( u \in \mathbb{F}_2^m \) denote by \( w_H(u) \) its Hamming weight, and for two vectors \( u, w \in \mathbb{F}_2^m \), \( d_H(u, w) \) is their Hamming distance. A binary code \( C \) of length \( m \) and size \( K \) over \( \mathbb{F}_2 \) will be denoted by \( (m, K) \) or \( (m, K, d) \), where \( d \) denotes its minimum Hamming distance. If \( C \) is also linear and \( k \) is its dimension, we denote the code by \( [m, k] \) or \( [m, k, d] \).

Let \( E_n \) be the set of all \( \binom{n}{2} \) edges as defined in (5.1), with a fixed order. For any set \( E \subseteq E_n \), let \( v_E \) be its characteristic vector of length \( \binom{n}{2} \) which is indexed.
by the edge set $E_n$ and every entry has value one if and only if the corresponding edge belongs to $E$. That is,

$$(v_E)_e = \begin{cases} 1, & e \in E \\ 0, & \text{otherwise} \end{cases}.$$

The construction of $T$-$(n,M,d)$ code over trees will be as follows.

**Construction 13.** For all $n \geq 1$ let $C$ be a binary code $n \choose 2$, $K$, $2d - 1$. Then, the code $C_{T_3}$ is defined by

$$(v_E) = \{T \in T(n) | v_E \in C\}.$$

**Theorem 68.** The code $C_{T_3}$ is a $T$-$(n,M,d)$ code over trees.

**Proof.** By Theorem 35, a code over trees $C_T$ with parameters $T$-$(n,M)$ has minimum distance $d$ if and only if $C_T$ can correct any $d - 1$ edge erasures. Notice also that since $C$ is a code with Hamming distance $2d - 1$, it can correct at most any $d - 1$ substitutions.

Let $T = (V,E)$ be a codeword-tree of $C_{T_3}$ with its binary edge-vector $v_E$. Suppose that $T$ experienced at most $d - 1$ edge erasures, generating a new forest $F$ with the edge set $E'$. Since $E' \subseteq E$ and $|E'| \geq |E| - (d - 1)$, it holds that $d_H(v_{E'}, v_E) \leq d - 1$ and the vector $v_E$ can be corrected using a decoder of $C$. \qed

The next corollary summarizes the result of this construction.

**Corollary 13.** For positive integer $n$ and fixed $d$, $A(n,d) = \Omega(n^{n - 2d})$ and the redundancy is $r(n,d) \leq (d - 1) \log(n) + O(1)$.

**Proof.** Applying BCH codes (see Chapter 5.6 in [27]) in Construction 13 for all $n \geq 1$, linear codes $n \choose 2$, $k$, $2d - 1$ are used with redundancy

$$r = (d - 1) \log\left(\frac{n}{2}\right) + O(1) = 2(d - 1) \log(n) + O(1)$$

redundancy bits. The $2^r$ cossets of the $C$ codes are also binary $n \choose 2$, $2^k$, $2d - 1$ codes. Note that each tree $T$ from $T(n)$ can be mapped by Construction 13 to exactly one of these cossets. Thus, by the pigeonhole principle, there exists a code $C_{T_3}$ of cardinality at least

$$\frac{n^{n - 2}}{2^{2(d - 1) \log(n) + O(1)}} = \frac{n^{n - 2}}{\alpha n^{2d - 2}} = \frac{1}{\alpha} n^{n - 2d},$$

for some constant $\alpha$. Thus, we also deduce that

$$r(n,d) \leq 2(d - 1) \log(n) + O(1).$$

\qed
Remark 1. We note that the use of BCH codes can be changed to any linear codes. In fact, it is possible to use in Construction 13 a code correcting \(d - 1\) asymmetric errors. However, we chose to use symmetric error-correcting codes since the use of asymmetric error-correcting codes does not improve the asymptotic result and in order to derive the result in Corollary 13 we needed linear codes.

In this section we showed a family of codes with \(\Omega(n^{n-2d})\) codeword-trees where \(d \leq n/2\). Next we show a construction of codes over trees with \(\Omega(n^2)\) codeword-trees where \(d\) is almost \(3n/4\).

5.8.4 A Construction of \(T(n, n-\frac{1}{2} \cdot \lfloor \frac{n-1}{m} \rfloor, \lfloor \frac{3n}{4} \rfloor - \lfloor \frac{3n}{2m} \rfloor - 2)\) Codes

In this section, for a prime \(n\), we show a construction of \(T(n, n-\frac{1}{2} \cdot \lfloor \frac{n-1}{m} \rfloor, \lfloor \frac{3n}{4} \rfloor - \lfloor \frac{3n}{2m} \rfloor - 2)\) codes, where \(m\) is a positive integer such that \(3 \leq m \leq n - 1\). By Corollary 13, \(A(n, d) = \Omega(n^{n-2d})\) where \(d \leq n/2\). Here we extend this result by showing that for \(d\) approaching \(3n/4\), there exists a code with \(\Omega(n^2)\) codeword-trees. First, several definitions are presented.

A two-star tree over \(n\) nodes is a tree which has exactly \(n - 2\) leaves. For a prime \(n\) and integers \(s, t \in [n]\) where \(t \neq 0\), denote the following two edge sets

\[ E_{s,t}^{(+) = \{ (v_{s}, v_{(s+t)}_{n}) | 1 \leq i \leq \frac{n+1}{2} \} \}, \]
\[ E_{s,t}^{(-} = \{ (v_{(s+n+1)_{n}}, v_{(s+n+2j+1)_{n}}) | 1 \leq j \leq \frac{n-3}{2} \}. \]

Denote by \(T_{s,t} = (V_{n}, E_{s,t})\) the two-star tree with the edge set

\[ E_{s,t} = E_{s,t}^{(+) \cup E_{s,t}^{(-}}. \]

It is possible to verify that indeed according to this definition \(T_{s,t}\) is well defined and is a two-star tree. Furthermore, it will be shown in Theorem 71 that each pair \((s, t)\) defines a unique tree \(T_{s,t}\). The nodes \(v_{s}\) and \(v_{(s+n+1)_{n}}\) are called the central nodes of \(T_{s,t}\). Also note that

\[ \deg(v_{s}) = \frac{n+1}{2}, \quad \deg(v_{(s+n+1)_{n}}) = \frac{n-1}{2}. \]  \hspace{1cm} (5.22)

In Fig. 5.6 we illustrate a two-star tree \(T_{s,t}\).

Example 17. In Fig. 5.6 we demonstrate a two-star tree. The nodes are marked by numbers \(i \in [n]\) instead of nodes \(v_{i}\). Note that by the definition of \(E_{s,t}^{(-}\), the node marked by \((s-t)_{n}\) is exactly the node marked by \((s+n+2j+1)_{n}\), where \(j = \frac{n-3}{2}\).

For a prime \(n\) and an integer \(1 \leq t \leq \lfloor \frac{n-1}{m} \rfloor\), where \(3 \leq m \leq n - 1\) and \(\alpha \in \{ \frac{n+1}{2}, \frac{n-1}{2} \}\), denote by \(W(n, t, \alpha)\) the set

\[ W(n, t, \alpha) = \{ (t)_{n}, (2t)_{n}, (3t)_{n}, \ldots, (\alpha t)_{n} \}. \]  \hspace{1cm} (5.23)

First we state the following claim.
Claim 32. For any two positive real numbers $a, b$ such that $a < b$, the number of integers $j$ such that $a < j \leq b$ is at most $\lceil b - a \rceil$.

The following lemma is now presented.

Lemma 69. Let $n$ be a prime number, $\alpha = \frac{n+1}{2}$, and $t_1, t_2$ be two distinct integers $1 \leq t_1, t_2 \leq \lfloor \frac{n-1}{m} \rfloor$. Then

$$|W(n, t_1, \alpha) \cap W(n, t_2, \alpha)| < \left\lfloor \frac{n}{4} \right\rfloor + \left\lfloor \frac{3n}{2m} \right\rfloor + 1.$$ 

Proof. It is sufficient to prove this claim for $t_1 = 1$, since all the other cases are proved by relabeling $t_1$ to 1 and $t_2$ to $t_2 - t_1 + 1$. In this case, $W(n, 1, \alpha) = \{1, 2, \ldots, \frac{n+1}{2}\} = \left\lfloor \frac{n+1}{2} \right\rfloor \setminus \{0\}$.

Thus, denote $t = t_2$ and since $0 \notin W(n, t, \alpha)$, it is sufficient to prove that for all $2 \leq t \leq \lfloor \frac{n-1}{m} \rfloor$,

$$\left\lfloor \frac{n+3}{2} \right\rfloor \cap W(n, t, \alpha) < \left\lfloor \frac{n}{4} \right\rfloor + \left\lfloor \frac{3n}{2m} \right\rfloor + 1.$$ 

For an integer $k$ such that $1 \leq k \leq \frac{n+1}{2m}t$, let $A_k$ be the set

$$A_k = \{(jt)_n \mid (k-1)\frac{n}{t} < j \leq k\frac{n}{t}\}.$$ 

Note that $jt = (k-1)n + (jt)_n$ and also

$$W(n, t, \alpha) = \bigcup_{k=1}^{\frac{n+1}{2m}t} A_k,$$

where all $A_k$’s are mutually disjoint. Moreover, for all $1 \leq k \leq \frac{n+1}{2m}t$,

$$|A_k| \leq \left\lfloor k\frac{n}{t} - (k-1)\frac{n}{t} \right\rfloor = \lfloor n/t \rfloor,$$
which holds due to Claim 32. Hence, 
\[
\left| A_k \cap \left[ \frac{n+3}{2} \right] \right| \leq \left| A_k \cap \left[ \frac{n-1}{2} \right] \right| + 2 \\
\leq \left| \left\{ \langle jt \rangle_n \mid (k-1) n t < j \leq (k-0.5) \frac{n}{t} \right\} \right| + 2 \\
\leq \left( k-0.5 \right) \frac{n}{t} - (k-1) \frac{n}{t} = \left[ \frac{n}{2t} \right] + 2,
\]
where (a) holds since for all \( \langle jt \rangle_n \in A_k \cap \left[ \frac{n-1}{2} \right] \) it holds that \( 0 < \langle jt \rangle_n < \frac{n-1}{2} \) and hence
\[
(k-1)n < jt = (k-1)n + \langle jt \rangle_n \\
< (k-1)n + \frac{n-1}{2} < (k-0.5)n.
\]
Equality (b) holds by Claim 32. Since \( t \leq \left\lfloor \frac{n-1}{m} \right\rfloor \), it is deduced that
\[
\left| W(n, t, \alpha) \cap \left[ \frac{n+3}{2} \right] \right| \leq \left| \bigcup_{k=1}^{\left\lfloor \frac{n+1}{n} \right\rfloor} A_k \cap \left[ \frac{n+3}{2} \right] \right| \\
\leq \frac{n+1}{2n} t \left( \left[ \frac{n}{2t} \right] + 2 \right) \\
\leq \frac{n+1}{2n} t \left( 3 \frac{n+1}{2} + 3 \right) \\
= \frac{n+1}{4} + \frac{3n+1}{2 n t} \\
\leq \frac{n+1}{4} + \frac{3n+1}{2 n m} \\
= \frac{n+1}{4} + \frac{3n^2-1}{2 nm} \\
< \left[ \frac{n}{4} \right] + \left[ \frac{3n}{2m} \right] + 1.
\]

Note that this lemma holds also for \( \alpha = \frac{n-1}{2} \). We state the following corollary which is derived directly from Lemma 32.

**Corollary 14.** Assume that \( W \) is a subset of one of the sets \( W(n, t, \alpha) \), where \( 1 \leq t \leq \left\lfloor \frac{n-1}{m} \right\rfloor \) and \( \alpha = \frac{n+1}{2} \). If \( |W| \geq \left[ \frac{n}{4} \right] + \left[ \frac{3n}{2m} \right] + 1 \), then the value of \( t \) can be uniquely determined.

This corollary holds also for \( \alpha = \frac{n-1}{2} \). We proceed by introducing several more definitions. For all \( 1 \leq t \leq \left\lfloor \frac{n-1}{m} \right\rfloor \) denote the following set
\[
B_t = \left\{ \left\langle \frac{n+1}{2} \cdot i \right\rangle_n \mid i \in [n], i \text{ is odd} \right\}
\]
and the set $A_{n,m}$ to be

$$A_{n,m} = \{ (s,t) \mid s \in B_t, 1 \leq t \leq \left\lfloor \frac{n-1}{m} \right\rfloor \}.$$  

Note that for every fixed $1 \leq t \leq \left\lfloor \frac{n-1}{m} \right\rfloor$, it holds that $|B_t| = \frac{n-1}{2}$. Thus,

$$|A_{n,m}| = \frac{n-1}{2} \cdot \left\lfloor \frac{n-1}{m} \right\rfloor. \quad (5.24)$$

Next, the following lemma is presented.

**Lemma 70.** For any $a \in [n]$, it holds that $(a - \frac{n+1}{2} t)_n \in A_{n,m}$ if and only if $(a,t) \notin A_{n,m}$.

**Proof.** If $(a - \frac{n+1}{2} t)_n \in A_{n,m}$ then $(a - \frac{n+1}{2} t)_n \in B_t$. Therefore, there is an odd $i \in [n]$ such that

$$\left( a - \frac{n+1}{2} t \right)_n = \left( \frac{n+1}{2} it \right)_n.$$  

Thus, $a = (\frac{n+1}{2}(i+1)t)_n$ when $i+1$ is even. Therefore, $a \notin B_t$ which leads to $(a,t) \notin A_{n,m}$. The opposite direction is proved similarly. \qed

The construction of a $\mathcal{T}$-$(n, \frac{n-1}{2} \cdot \left\lfloor \frac{n-1}{m} \right\rfloor, \left\lfloor \frac{3n}{4} \right\rfloor - \left\lceil \frac{3n}{2m} \right\rceil - 2)$ code will be as follows.

**Construction 14.** For a prime $n \geq 3$ let $C_{T_4}$ be the following code over trees

$$C_{T_4} = \{ T_{s,t} = (V_n, E_{s,t}) \mid (s,t) \in A_{n,m} \}.$$  

**Theorem 71.** The code $C_{T_4}$ is a $\mathcal{T}$-$(n, \frac{n-1}{2} \cdot \left\lfloor \frac{n-1}{m} \right\rfloor, \left\lfloor \frac{3n}{4} \right\rfloor - \left\lceil \frac{3n}{2m} \right\rceil - 2)$ code over trees.

**Proof.** First, it is deduced above in (5.24) that $|A_{n,m}| = \frac{n-1}{2} \cdot \left\lfloor \frac{n-1}{m} \right\rfloor$. We now prove that

$$|C_{T_4}| = |A_{n,m}| = \frac{n-1}{2} \cdot \left\lfloor \frac{n-1}{m} \right\rfloor.$$  

It is clear that $|C_{T_4}| \leq |A_{n,m}|$ and assume in the contrary that $|C_{T_4}| < |A_{n,m}|$. Thus, there are two distinct pairs $(s,t), (s',t') \in A_{n,m}$ such that $T_{s,t} = T_{s',t'}$, which implies that the central nodes of $T_{s,t}$ and $T_{s',t'}$ are identical. Since $\text{deg}(s) = \text{deg}(s')$, the nodes $v_s$ and $v_{s'}$ represent the same center node, so it is deduced that $s = s'$. From that, by the definition of the second central node, it is immediately implied that $t = t'$ which results with a contradiction.

Next, we show that $d = \left\lfloor \frac{3n}{4} \right\rfloor - \left\lceil \frac{3n}{2m} \right\rceil - 2$ by showing that it is possible to correct $\rho = d - 1$ edge erasures due to Theorem 35. Assume that $\rho$ edges are erased in a tree $T_{s,t} \in C_{T_4}$. We separate the proof for two cases.
Case 1: after the erasure, both central nodes have degree of at least two, and will be denoted by \( v_a \) and \( v_b \). If \( a = s \) and \( b = (s + \frac{n+1}{2}t)_n \), then

\[
\langle (a - b) \cdot 2 \rangle_n = \langle (s - (s + \frac{n+1}{2}t)) \cdot 2 \rangle_n = \langle -t \rangle_n.
\]

Similarly, if \( a = (s + \frac{n+1}{2}t)_n \) and \( b = s \), then

\[
\langle (a - b) \cdot 2 \rangle_n = \langle ((s + \frac{n+1}{2}t) - s) \cdot 2 \rangle_n = t.
\]

Since \( t \leq \lfloor \frac{n-1}{m} \rfloor \), it is deduced that \( \lfloor \frac{n-1}{m} \rfloor < \langle -t \rangle_n \leq n - 1 \), so only one of these options is valid and \( t \) is easily determined. Moreover, it is now determined which one of the values \( a \) or \( b \) is equal to \( s \), and thus, \( T_{s,t} \) is corrected.

Case 2: after the erasure, one of the central nodes has degree of at most one. Denote by \( v_a \) the central node with degree of at least two. Let \( \alpha \) be a number such that if \( a = s \) then \( \alpha = \frac{n+1}{2} \) and if \( a = (s + \frac{n+1}{2}t)_n \) then \( \alpha = \frac{n-1}{2} \). Note that since \( \rho \) edges were erased, \( v_a \) has degree of at least

\[
(n - 1) - \rho - 1 = (n - 1) - \left( \frac{3n}{4} \right) - \left( \frac{3n}{2m} - 3 \right) - 1 = \left\lfloor \frac{n}{4} \right\rfloor + \left\lfloor \frac{3n}{2m} \right\rfloor + 1.
\]

Thus, there are integers \( i_1, i_2, \ldots, i_{(n-2)-\rho} \in [n] \) such that the edge set

\[
E = \{ \langle v_{a\alpha}, v_{(a+i_j)t} \rangle_n | 1 \leq j \leq (n - 2) - \rho \}
\]

consists of all the edges connected to \( v_a \) and were not erased. Let \( W(n,t,\alpha) \) be the set defined in (5.23), and let \( W \) be the set

\[
W = \left\{ \langle ij,t \rangle_n \in W(n,t,\alpha) | 1 \leq j \leq (n - 2) - \rho, \langle v_{a\alpha}, v_{(a+i_j)t} \rangle_n \in E \right\}.
\]

Since \( |W| = (n-2)-\rho = \left\lfloor \frac{n}{4} \right\rfloor + \left\lfloor \frac{3n}{2m} \right\rfloor + 1 \), by Corollary 14, the value of \( t \) is uniquely determined. Therefore, the codeword-tree \( T_{s,t} \) is either \( T_{a,t} \) or \( T_{(a - \frac{n+1}{2}t)_n,t} \). By Lemma 70, it holds that \( \langle a - \frac{n+1}{2}t \rangle_n \in A_{n,m} \) if and only if \( (a, t) \notin A_{n,m} \). Thus, \( T_{a,t} \in \mathcal{C}_T \) if and only if \( T_{(a - \frac{n+1}{2}t)_n,t} \notin \mathcal{C}_T \), and by finding either \( T_{a,t} \) or \( T_{(a - \frac{n+1}{2}t)_n,t} \) in \( \mathcal{C}_T \) we find the codeword-tree \( T_{s,t} \).

Note that according to Theorem 71 it is possible to construct codes of cardinality \( \Omega(n^2) \), while the minimum distance \( d \) approaches \( \lfloor 3n/4 \rfloor \) and \( n \) is a prime number. In Theorem 50 we showed that \( A(n, n - 3) = \Theta(n^2) \), while from Theorem 71 \( A(n, d) = \Omega(n^2) \), when \( d \) approaches \( \lfloor 3n/4 \rfloor \) and \( n \) is prime. Thus, it is interesting to study the values of \( d \) for such that \( A(n, d) = \Theta(n^2) \).
5.9 Conclusion

In this paper, we initiated the study of codes over trees over the tree distance. Upper bounds on such codes were presented together with specific code construction for several parameters of the number of nodes and minimum tree distance. For the tree ball of trees, it was shown that the star tree reaches the smallest size, while the maximum is achieved for the path tree. This guarantees that for a fixed value of $t$, the size of every ball of a tree is lower, upper-bounded from below, above by $\Omega(n^{2t})$, $O(n^{3t})$, respectively. Furthermore, it was also shown that the average size of the ball is $\Theta(n^{2.5t})$. We also showed that optimal codes over trees ranged between $O(n^{n-d-1})$ and $\Omega(n^{n-2d})$.

While the results in the paper provide a significant contribution in the area of codes over trees, there are still several interesting problems which are left open. Some of them are summarized as follows.

1. Improve the lower and upper bounds on the size of codes over trees, that is, the value of $A(n,d)$.

2. Find an optimal construction for $d = n - 3$.

3. Study codes over trees under different metrics such as the tree edit distance.

4. Study the problem of reconstructing trees based upon several forests in the forest ball of trees; for more details see [7].

Acknowledgment

The authors would like to thank the associate editor Amin Gohari and the three anonymous reviewers for their very helpful comments.

Appendix A

Lemma \[53\] For a positive integer $n$ it holds that

$$
\sum_{T \in T(n)} V_T(n, 1) = \sum_{F \in F(n, 2)} (V_F(n, 1))^2 - (n - 2)n^{n-2}.
$$

156
Proof. The following sequence of equalities will be explained below,

\[
\sum_{T \in T(n)} V_T(n, 1) = \\
\overset{(a)}{=} \sum_{T \in T(n)} \left( \sum_{F \in P_T(n, 1)} (V_F(n, 1) - 1) + 1 \right) \\
= \sum_{T \in T(n)} \sum_{F \in P_T(n, 1)} V_F(n, 1) - \left( \sum_{T \in T(n)} \sum_{F \in P_T(n, 1)} 1 \right) + n^{n-2} \\
\overset{(b)}{=} \sum_{T \in T(n)} \sum_{F \in P_T(n, 1)} V_F(n, 1) - (n-1)n^{n-2} + n^{n-2} \\
\overset{(c)}{=} \sum_{F \in F(n, 2)} \sum_{T \in B_F(n, 1)} V_F(n, 1) - (n-1)n^{n-2} + n^{n-2} \\
= \sum_{F \in F(n, 2)} (V_F(n, 1))^2 - (n-2)n^{n-2}.
\]

In equality (a) we explain why

\[
V_T(n, 1) - 1 = |B_T(n, 1) \setminus \{T\}| = \sum_{F \in P_T(n, 1)} (V_F(n, 1) - 1).
\]

Note that for all \(T, T' \in T(n)\) such that \(d_T(T, T') = 1\), there exists exactly one forest \(F \in F(n, 2)\) such that \(F \in P_T(n, 1) \cap P_{T'}(n, 1)\). Thus, each \(T' \in B_T(n, 1) \setminus \{T\}\) can be generated from \(T\) uniquely by removing and adding exactly one edge. Equivalently, each such a tree is counted by adding an edge to a forest \(F \in P_T(n, 1)\). By doing so for all forests in \(P_T(n, 1)\), while subtracting 1 for the tree \(T\), equality (a) holds. In equality (b) it is deduced that

\[
\sum_{T \in T(n)} \sum_{F \in P_T(n, 1)} 1 = (n-1)n^{n-2}.
\]

Lastly, in equality (c), by taking pairs of trees and forests, the order of summation is changed,

\[
\sum_{T \in T(n)} \sum_{F \in P_T(n, 1)} 1 = \sum_{F \in F(n, 2)} \sum_{T \in B_F(n, 1)} 1.
\]

\[\square\]

Appendix B

Theorem 56. For any \(T \in T(n)\) it holds that

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) = n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in P_T(n, t)} i_0 i_1 \cdots i_t.
\]
Proof. By definition, for \( t \geq 1 \), \( S_T(n, t) = V_T(n, t) - V_T(n, t - 1) \). Thus,

\[
\sum_{i=0}^{t} \binom{n-1-t+i}{i} S_T(n, t-i) = \\
\sum_{i=0}^{t-1} \binom{n-1-t+i}{i} \left( V_T(n, t-i) - V_T(n, t-1-i) \right) \\
+ \binom{n-1}{t} V_T(n, 0) = V_T(n, t) \\
+ \sum_{i=1}^{t} V_T(n, t-i) \left( \binom{n-1-t+i}{i} - \binom{n-2-t+i}{i-1} \right) \\
= V_T(n, t) + \sum_{i=1}^{t} V_T(n, t-i) \binom{n-2-t+i}{i} \\
= \sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i),
\]

where \((a)\) holds by the identity \( \binom{n}{k} + \binom{n+1}{k} = \binom{n+1}{k+1} \). Using the result of Corollary \([\square]\) we conclude the proof. \( \square \)

Appendix C

Lemma 57 For any positive integer \( \alpha \), if

\[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n, t-i) = \Omega(n^{\alpha t}),
\]

and \( V_T(n, 0) = 1 \), then \( V_T(n, t) = \Omega(n^{\alpha t}) \).

Proof. This lemma is proved by induction on \( t \).

Base: for \( t = 0 \), \( V_T(n, 0) = n^0 = 1 \) which is true by the definition.
Inductive Step: suppose that the lemma holds for all $0 \leq t' \leq t - 1$. Thus,

\[ \Omega(n^{\alpha t}) = \sum_{i=0}^{t} \binom{n}{i} V_T(n, t - i) \]

\[ = V_T(n, t) + \sum_{i=1}^{t} \binom{n}{i} V_T(n, t - i) \]

\[ = V_T(n, t) + \sum_{i=1}^{t} \binom{n}{i} \Omega(n^{\alpha(t-i)}) \]

\[ = V_T(n, t) + \sum_{i=1}^{t} \Omega(n^{i}) \Omega(n^{\alpha(t-i)}) \]

\[ = V_T(n, t) + \Omega(n^{\alpha(t-1)+1}). \]

Therefore we deduce that

\[ V_T(n, t) = \Omega(n^{\alpha t}) - \Omega(n^{\alpha(t-1)+1}) = \Omega(n^{\alpha t}). \]

Appendix D

Claim 30. For a positive integer $n$ and a fixed $t$ it holds that

\[ \sum_{i=1}^{n-1} \binom{n}{i} i^i (n-i)^{n-i} \Theta(t^{i/2}) = \Theta(t^{i/2}) \sum_{i=1}^{n-1} \binom{n}{i} i^i (n-i)^{n-i}. \]

Proof. The upper bound is derived immediately,

\[ \sum_{i=1}^{n-1} \binom{n}{i} i^i (n-i)^{n-i} \Theta(t^{i/2}) = O(t^{i/2}) \sum_{i=1}^{n-1} \binom{n}{i} i^i (n-i)^{n-i}. \]

Next, the lower bound is proved by,

\[ \sum_{i=1}^{n-1} \binom{n}{i} i^i (n-i)^{n-i} \Omega(t^{i/2}) \]

\[ \geq \sum_{i=\lceil n/2 \rceil}^{n-1} \binom{n}{i} i^i (n-i)^{n-i} \Omega(t^{i/2}) \]

\[ = \Omega(t^{i/2}) \sum_{i=\lceil n/2 \rceil}^{n-1} \binom{n}{i} i^i (n-i)^{n-i} \]

\[ = \Omega(t^{i/2}) \sum_{i=1}^{n-1} \binom{n}{i} i^i (n-i)^{n-i}. \]
Appendix E

Theorem 61. The size of the sphere for a star satisfies
\[ S^*(n, t) = \binom{n-1}{t}(n-1)^{t-1}(n-t-1), \]
and the size of the tree ball of trees for a star satisfies
\[ V^*(n, t) = \sum_{j=0}^{t} \binom{n-1}{j}(n-1)^{j-1}(n-j-1). \]

Proof. Let \( T \in \mathbf{T}(n) \) be a star tree, and denote the function
\[ H(n, t) = \binom{n-1}{t}(n-1)^{t-1}(n-t-1). \]
We say that \( \frac{d}{dn}(f(n)) \) is the derivative of \( f(n) \) with respect to \( n \). Thus,
\[
\sum_{i=0}^{t} \binom{n-1-t+i}{i} H(n, t-i) \\
= \sum_{i=0}^{t} \binom{n-1-t+i}{i}(n-1)^{t-1-i}(n-t-1+i) \\
= \binom{n-1}{t} \sum_{i=0}^{t} \binom{t}{i}(n-1)^{t-1-i}(n-t-1+i) \\
= \binom{n-1}{t} \left( \sum_{i=0}^{t} \binom{t}{i}(n-1)^{t-1-i}(n-1) \\
- \sum_{i=0}^{t} \binom{t}{i}(n-1)^{t-1-i}(t-i) \right) \\
= \binom{n-1}{t} \left( \sum_{i=0}^{t} \binom{t}{i}(n-1)^{t-1-i} - \frac{d}{dn} \left( \sum_{i=0}^{t} \binom{t}{i}(n-1)^{t-i} \right) \right) \\
= \binom{n-1}{t} \left( n^t - tn^{t-1} \right) = \binom{n-1}{t} n^{t-1}(n-t) \\
= n^{t-1} \sum_{(i_0, i_1, \ldots, i_t) \in P^*_T(n, t)} i_0i_1 \cdots i_t,
\]
where (a) holds by known formula
\[
\left( \begin{array}{c} a \end{array} \right) \left( \begin{array}{c} b \end{array} \right) = \left( \begin{array}{c} a - (b - c) \end{array} \right) \left( \begin{array}{c} a \\ b - c \end{array} \right) = \left( \begin{array}{c} a \\ b \\ c \end{array} \right)
\]
160
(i.e. \(a = n - 1\), \(b = t\), and \(c = i\)), and \((b)\) holds by the binomial theorem, which is,

\[
\sum_{i=0}^{t} \binom{t}{i} (n - 1)^{t-i} = (n - 1 + 1)^t = n^t.
\]

Equality \((c)\) holds due to (5.17). Thus, by Corollary 3, it is deduced that \(S^*(n, t) = H(n, t)\). Next,

\[
V^*(n, t) = \sum_{j=0}^{t} \binom{n-1}{j} (n - 1)^{j-1}(n - j - 1),
\]

which is derived by the fact that for every \(T \in T(n)\),

\[
V_T(n, t) = \sum_{i=0}^{t} S_T(n, i).
\]

\[
\square
\]

Appendix F

Claim 31. The following properties hold

1. It holds that

\[
\sum_{(c_0, \ldots, c_t) \in P_T(n, t; v)} c_\ell \cdots c_t = \sum_{(c_0, \ldots, c_t) \in A_T(n, t; v_\ell, v_y)} c_\ell \cdots c_t + \sum_{(c_0, \ldots, c_t) \in P_T(n, t; v_{\ell+1}(v_y))} c_\ell \cdots c_t.
\]

2. If \(v_x\) is not in \(v_\ell\) then,

\[
\sum_{(c_0, \ldots, c_t) \in A_T(n-1, t; v_\ell, v_y)} c_\ell \cdots c_t = \sum_{(c'_0, \ldots, c'_t) \in A_T(n, t; v_{\ell+1}(v_y))} c'_\ell \cdots c'_t.
\]

3. If \(v_x\) is not in \(v_\ell\) then,

\[
\sum_{(c_0, \ldots, c_t) \in P_T(n, t; v_{\ell+1}(v_y))} c_\ell \cdots c_t = \sum_{(c'_0, \ldots, c'_t) \in P_T(n-1, t; v_{\ell+1}(v_y))} (c'_\ell + 1)c'_{\ell+1} \cdots c'_t + \sum_{(c_0, \ldots, c_{\ell-1}, c_{\ell+1}, \ldots, c_t) \in P_T(n, t; v_{\ell+1}(v_y))} 1 \cdot c_{\ell+1} \cdots c_t.
\]
4. If \( v_x = v_{i+1} \) then

\[
\sum_{(c_0, \ldots, c_{i+1}) \in P_T(n, t; v_{i+1}(v_x))} c_{i+1} \cdot \cdot \cdot c_t
\]

\[
= \sum_{(c_0', \ldots, c_{i+1}') \in P_T(n-1, t; v_{i+1}(v_y))} c_{i+1}' \cdot \cdot \cdot c_t
\]

\[
+ \sum_{(c_0, \ldots, c_{i+1}, c_{i+2}, \ldots, c_t) \in P_T(n, t; v_{i+1}(v_x))} c_{i+1} \cdot \cdot \cdot c_t.
\]

Proof. 1. By definition of the set \( A_T(n, t; v_\ell, v_y) \) it holds that \( A_T(n, t; v_\ell, v_y) \subseteq P_T(n, t; v_\ell) \). Moreover,

\[
\sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_\ell) \setminus A_T(n, t; v_\ell, v_y)} c_{i+1} \cdot \cdot \cdot c_t
\]

\[
= \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_\ell+1(v_y))} c_{i+1} \cdot \cdot \cdot c_t.
\]

and the proof is concluded.

2. Again, \( A_T(n, t; v_\ell, v_x) \subseteq P_T(n, t; v_\ell) \). Since \( v_x \) is not in \( v_\ell \) it holds that \( v_x \) and \( v_y \) are always in the same connected component with respect to \( A_T(n, t; v_\ell, v_x) \), and thus, \( |A_T(n, t; v_\ell, v_x)| = |A_T(n-1, t; v_\ell, v_y)| \). Moreover, since there is an index \( j \in [\ell] \) such that \( v_x, v_y \in C_j \) in \( T \), it holds that \( (c_0, \ldots, c_j, \ldots, c_t) \in A_T(n, t; v_\ell, v_x) \) if and only if \( (c_0, \ldots, (c_j-1), \ldots, c_t) \in A_T(n-1, t; v_\ell, v_y) \). Hence, this difference does not affect the equality, which concludes this proof.

3. Assume that \( v_x \) and \( v_y \) are in the same connected component \( C_\ell \) with respect to \( T \). In this case since \( v_x, v_y \in C_\ell \), it holds that \( (c_0, \ldots, c_{i+1}, \ldots, c_t) \in P_T(n, t; v_{i+1}(v_x)) \) if and only if \( (c_0, \ldots, c_{i+1}, \ldots, c_t) \in P_T(n-1, t; v_{i+1}(v_y)) \). Thus, the following expression

\[
\sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{i+1}(v_x))} (c_0' + 1) c_1' \cdot \cdot \cdot c_t'
\]

corresponds to all cases where the edge \( (v_x, v_y) \) was not removed, and

\[
\sum_{(c_0, \ldots, c_{i+1}, c_{i+2}, \ldots, c_t) \in P_T(n, t; v_{i+1}(v_x))} 1 \cdot c_{i+1} \cdot \cdot \cdot c_t,
\]

corresponds to all cases where the edge \( (v_x, v_y) \) was removed. Hence, the sum of the two expressions equals to

\[
\sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{i+1}(v_x))} c_{i+1} \cdot \cdot \cdot c_t.
\]
4. Assume that \( v_x \) and \( v_y \) are in the same connected component \( C_{\ell-1} \) with respect to \( T \). Since \( v_x, v_y \in C_{\ell-1} \), it holds that \((c_0, \ldots, c_{\ell-1}, c_\ell) \in P_T(n, t; v_\ell(v_x)) \) if and only if \((c_0, \ldots, c_{\ell-1} - 1, c_\ell) \in P_T((n-1), t; v_\ell(v_y)) \).

Thus, the following expression

\[
\sum_{(c'_0, c'_1, \ldots, c'_\ell) \in P_{T_1}(n-1, t; v_\ell(v_y))} c'_\ell \cdots c'_t,
\]

corresponds to all cases where the edge \( (v_x, v_y) \) was not removed, and

\[
\sum_{(c_0, \ldots, c_{\ell-2}, c_\ell, \ldots, c_t) \in P_T(n, t; v_\ell(v_x))} c_\ell \cdots c_t.
\]

corresponds to all cases where the edge \( (v_x, v_y) \) was removed. Again we get that the sum of the two expressions equals to

\[
\sum_{(c_0, \ldots, c_t) \in P_T(n, t; v_\ell(v_x))} c_\ell \cdots c_t.
\]

\[\square\]

Appendix G

**Lemma 63.** If \( v_x \) is not in \( v_\ell \) then,

\[
f_T(n, t; v_\ell) = f_{T_1}(n-1, t; v_\ell) + f_{T_1}(n-1, t; v_{\ell+1}(v_y)) + f_{T_1}(n-1, t-1; v_\ell).
\]

If \( v_x \) is in \( v_\ell \), and without loss of generality \( v_x = v_{\ell-1} \), then

\[
f_T(n, t; v_\ell) = f_{T_1}(n-1, t; v_{\ell}(v_y)) + f_{T_1}(n-1, t-1; v_{\ell-1}).
\]

**Proof.** In this proof, it is assumed that \( v_y \) is not in \( v_\ell \), although the proof is valid also for this case, where by the definition \( f_{T_1}(n-1, t; v_{\ell+1}(v_y)) = 0 \). First we
prove the case where \( v_x \) is not in \( v_\ell \). In this case, we have that

\[
\begin{align*}
  f_{T_1}(n - 1, t; v_\ell) &+ f_{T_1}(n - 1, t; v_{\ell+1}(v_y)) \\
  \equiv & \sum_{(c_0', c_1', \ldots, c_t') \in P_{T_1}(n - 1, t; v_\ell)} c_t' \\
  + & \sum_{(c_0', c_1', \ldots, c_t') \in P_{T_1}(n - 1, t; v_{\ell+1}(v_y))} c_{t+1}' \cdots c_t' \\
  \equiv & \sum_{(c_0', c_1', \ldots, c_t') \in A_{T_1}(n - 1, t; v_\ell, v_y)} c_t' \\
  + & \sum_{(c_0', c_1', \ldots, c_t') \in P_{T_1}(n - 1, t; v_{\ell+1}(v_y))} c_{t+1}' \cdots c_t' \\
  = & \sum_{(c_0', c_1', \ldots, c_t') \in A_{T_1}(n - 1, t; v_\ell, v_y)} c_t' \\
  + & \sum_{(c_0', c_1', \ldots, c_t') \in P_{T_1}(n - 1, t; v_{\ell+1}(v_y))} (c_{t} + 1) \cdots c'_t
\end{align*}
\]

Equality (a) holds by definition of the function \( f_{T_1} \). Equality (b) holds due to Claim 31(a). Equality (c) holds by Claim 31(b) and (c). Equality (d) holds again due to Claim 31(a). Next we show that

\[
\begin{align*}
  f_{T_1}(n - 1, t - 1; v_\ell) & \\
  = & \sum_{(c_0', \ldots, c_{\ell-1}', c_{\ell+1}', \ldots, c'_t) \in P_{T_1}(n - 1, t - 1; v_\ell)} c_{t+1}' \cdots c_t' \\
  \equiv & \sum_{(c_0, \ldots, c_{\ell-1}, c_{\ell+1}, \ldots, c_t) \in P_T(n, t; v_{\ell+1}(v_y))} c_{t+1} \cdots c_t
\end{align*}
\]

164
where \((a)\) holds since \((c_0, \ldots, c_{\ell - 1}, c_{\ell + 1}, \ldots, c_t) \in P_{T_1}(n - 1, t - 1; v_{\ell})\) if and only if \((c_0, \ldots, c_{\ell - 1}, 1, c_{\ell + 1}, \ldots, c_t) \in P_T(n, t; v_{\ell})\). Thus,

\[
f_{T_1}(n - 1, t; v_{\ell}) + f_{T_1}(n - 1, t; v_{\ell + 1}(v_y)) + f_{T_1}(n - 1, t - 1; v_{\ell})
= \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell})} c_{\ell} \cdots c_t
- \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell + 1}(v_y))} 1 \cdot c_{\ell + 1} \cdots c_t
+ \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell + 1}(v_y))} 1 \cdot c_{\ell + 1} \cdots c_t
= \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell})} c_{\ell} \cdots c_t
= f_T(n, t; v_{\ell}).
\]

Similarly, if \(v_x\) is in \(v_{\ell}\) and \(v_x = v_{\ell - 1}\),

\[
f_{T_1}(n - 1, t; v_{\ell}(v_y)) + f_{T_1}(n - 1, t - 1; v_{\ell - 1})
= \left(\sum_{(c'_{0}, c'_{1}, \ldots, c'_{\ell}) \in P_{T_1}(n - 1, t; v_{\ell}(v_y))} c'_{\ell} \cdots c'_{t}\right)
+ \sum_{(c'_{0}, c'_{1}, \ldots, c'_{\ell - 1}, c_{\ell + 1}, \ldots, c_t) \in P_{T_1}(n - 1, t - 1; v_{\ell - 1})} c'_{\ell} \cdots c'_{t}
= \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell}(v_x))} c_{\ell} \cdots c_t
- \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell}(v_x))} c_{\ell} \cdots c_t
+ \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell}(v_x))} c_{\ell} \cdots c_t
= \sum_{(c_0, c_1, \ldots, c_t) \in P_T(n, t; v_{\ell}(v_x))} c_{\ell} \cdots c_t
= f_T(n, t; v_{\ell}(v_x)) = f_T(n, t; v_{\ell}),
\]

where equality \((a)\) holds by the definition of \(f_{T_1}\), equality \((b)\) holds due to Claim \(d\), and since \((c_0, \ldots, c_{\ell - 2}, c_{\ell}, \ldots, c_t) \in P_{T_1}(n - 1, t - 1; v_{\ell - 1})\) if and only if \((c_0, \ldots, c_{\ell - 2}, 1, c_{\ell}, \ldots, c_t) \in P_T(n, t; v_{\ell}(v_x))\).
Appendix H

Lemma 64. For any tree \( T \in T(n) \), \( n \geq 1 \) and a vector of \( 0 \leq \ell \leq t + 1 \leq n \) nodes \( v_\ell = (v_{i_0}, v_{i_1}, \ldots, v_{i_{\ell-1}}) \),

\[
f_T(n, t; v_\ell) \leq \binom{n + t - \ell}{2t + 1 - \ell}.
\]

Proof. Note that if \( \ell = t + 1 \) by the definition of \( f_T \)

\[
f_T(n, t; v_{t+1}) \leq \binom{n - 1}{t} = \binom{n + t - (t + 1)}{2t + 1 - (t + 1)}.
\]

As showed in (5.19), if \( n = t + 1 \), then

\[
f_T(n, n - 1; v_\ell) = 1 = \binom{n + (n - 1) - \ell}{2(n - 1) + 1 - \ell},
\]

so the lemma is correct for this two cases. Thus it is left to prove the cases where \( 0 \leq \ell < t + 1 < n \), and it will be shown by the induction on \( n \geq 1 \).

Base: immediately derived from (25).

Inductive Step: assume that for any tree \( T \in T(n - 1), n \geq 1 \) and a vector of \( 1 \leq \ell \leq t + 1 \leq n - 1 \) nodes \( v_\ell = (v_{i_0}, v_{i_1}, \ldots, v_{i_{\ell-1}}) \),

\[
f_T(n - 1, t; v_\ell) \leq \binom{n - 1 + t - \ell}{2t + 1 - \ell}.
\]

Let \( T \in T(n) \) and let \( v_x \) be a leaf connected to a node denoted by \( v_y \). Assume that \( T_1 \in T(n - 1) \) is the tree generated by removing \( v_x \) from \( T \). For two integers \( t \) and \( \ell \) such that \( 0 \leq \ell < t + 1 < n \), let \( v_\ell = (v_{i_0}, v_{i_1}, \ldots, v_{i_{\ell-1}}) \) be a vector of \( \ell \) nodes in \( T \). If \( v_x \) is not in \( v_\ell \), using Lemma 63 and the induction assumption, we deduce that

\[
f_T(n, t; v_\ell) = f_{T_1}(n - 1, t; v_\ell)
+ f_{T_1}(n - 1, t; v_{\ell+1}(v_y)) + f_{T_1}(n - 1, t - 1; v_\ell)
\leq \binom{n - 1 + t - \ell}{2t + 1 - \ell} + \binom{n - 1 + t - \ell - 1}{2t + 1 - \ell - 1}
+ \binom{n - 1 + t - 1 - \ell}{2t - 1 - \ell}
= \binom{n - 1 + t - \ell}{2t + 1 - \ell} + \binom{n - 1 + t - \ell}{2t - \ell}
= \binom{n + t - \ell}{2t + 1 - \ell},
\]

166
where each equality holds by the identity $\binom{n}{k} + \binom{n+1}{k} = \binom{n+1}{k+1}$. Similarly, if $v_x \in v_\ell$, and without loss of generality $v_x = v_{\ell-1}$, then

$$f_T(n, t; v_\ell) = f_{T_1}(n - 1, t; v_\ell(v_y)) + f_{T_1}(n - 1, t - 1; v_{\ell-1})$$

$$\leq \left( \frac{n - 1 + t - \ell}{2t + 1 - \ell} \right) + \left( \frac{n - 1 + t - 1 - \ell + 1}{2t - 1 - \ell + 1} \right)$$

$$= \left( \frac{n + t - \ell}{2t + 1 - \ell} \right).$$

\qed
Bibliography


168


Part II

Unpublished Published Journal Papers
Chapter 6

Almost Optimal Construction of Functional Batch Codes Using Hadamard Codes

Lev Yohananov and Eitan Yaakobi

Abstract

A functional $k$-batch code of dimension $s$ consists of $n$ servers storing linear combinations of $s$ linearly independent information bits. Any multiset request of size $k$ of linear combinations (or requests) of the information bits can be recovered by $k$ disjoint subsets of the servers. The goal under this paradigm is to find the minimum number of servers for given values of $s$ and $k$. A recent conjecture states that for any $k = 2^s - 1$ requests the optimal solution requires $2^s - 1$ servers. This conjecture is verified for $s \leq 5$ but previous work could only show that codes with $n = 2^s - 1$ servers can support a solution for $k = 2^{s-2} + 2^{s-4} + \left\lfloor \frac{2^{s/2}}{\sqrt{3}} \right\rfloor$ requests. This paper reduces this gap and shows the existence of codes for $k = \left\lfloor \frac{5}{6} 2^s - 1 \right\rfloor - s$ requests with the same number of servers. Another construction in the paper provides a code with $n = 2^{s+1} - 2$ servers and $k = 2^s$ requests, which is an optimal result. These constructions are mainly based on Hadamard codes and equivalently provide constructions for parallel Random I/O (RIO) codes.

6.1 Introduction

Motivated by several applications for load-balancing in storage and cryptographic protocols, batch codes were first proposed by Ishai et al. [6]. A batch code encodes a length-$s$ string $x$ into $n$ strings, where each string corresponds to a server, such that each batch request of $k$ different bits (and more generally symbols) from $x$ can be decoded by reading at most $t$ bits from every server. This decoding
process corresponds to the case of a single-user. There is an extended variant for batch codes [6] which is intended for a multi-user application instead of a single-user setting, known as the multiset batch codes. Such codes have \( k \) different users and each requests a single data item. Thus, the \( k \) requests can be represented as a multiset of the bits since the requests of different users may be the same, and each server can be accessed by at most one user.

A special case of multiset batch codes, referred as primitive batch codes, is when each server contains only one bit. The goal of this model is to find, for given \( s \) and \( k \), the smallest \( n \) such that a primitive batch code exists. This problem was considered in several papers; see e.g. [1, 2, 6, 7, 11]. By setting the requests to be a multiset of linear combinations of the \( s \) information bits, a batch code is generalized into a functional batch code [14]. Again, given \( s \) and \( k \), the goal is to find the smallest \( n \) for which a functional \( k \)-batch code exists.

Mathematically speaking, an \( FB-(n,s,k) \) functional \( k \)-batch code (and in short \( FB-(n,s,k) \) code) of dimension \( s \) consists of \( n \) servers storing linear combinations of \( s \) linearly independent information bits. Any multiset of size \( k \) of linear combinations from the linearly independent information bits, can be recovered by \( k \) disjoint subsets of servers. If all the \( k \) linear combinations are the same, then the servers form an \( FP-(n,s,k) \) functional \( k \)-Private Information Retrieval (PIR) code (and in short \( FP-(n,s,k) \) code). Clearly, an \( FP-(n,s,k) \) code is a special case of an \( FB-(n,s,k) \) code. It was shown that functional \( k \)-batch codes are equivalent to the so-called linear parallel random I/O (RIO) codes, where RIO codes were introduced by Sharon and Alrod [8], and their parallel variation was studied in [9, 10]. Therefore, all the results for functional \( k \)-batch codes of this paper hold also for parallel RIO codes. If all the \( k \) linear combinations are of a single information bit (rather than linear combinations of information bits), then the servers form an \( B-(n,s,k) \) \( k \)-batch code (and in short \( B-(n,s,k) \) code).

The value \( FP(s,k), B(s,k), FB(s,k) \) is defined to be the minimum number of servers required for the existence of an \( FP-(n,s,k) \), \( B-(n,s,k) \), \( FB-(n,s,k) \) code, respectively. Several upper and lower bounds can be found in [14] on these values. Wang et al. [12] showed that for \( k = 2^{s-1} \), the length of an optimal \( k \)-batch code is \( 2^s - 1 \), that is, \( B(s,k = 2^{s-1}) = 2^s - 1 \). They also showed a recursive decoding algorithm. It was conjectured in [14] that for the same value of \( k \), the length of an optimal functional batch code is \( 2^s - 1 \), that is, \( FB(s,k = 2^{s-1}) = 2^s - 1 \). Indeed, in [13] this conjecture was proven for \( s = 3, 4 \), and in [14], by using a computer search, it was verified also for \( s = 5 \). However, the best-known result for \( s > 5 \) only provides a construction of \( FB-(2^s - 1, s, 2^{s-2} + 2^{s-4} + \left\lceil \frac{2^{s/2}}{\sqrt{2^s}} \right\rceil \) codes [14]. This paper significantly improves this result and reduces the gap between the conjecture statement and the best-known construction. In particular, a construction of \( FB-(2^s - 1, s, \left\lceil \frac{5}{8} \cdot 2^{s-1} \right\rceil - s) \) codes is given. To obtain this important result, we first show an existence of \( FB-(2^s - 1, s, \left\lceil \frac{3}{4} \cdot 2^{s-1} \right\rceil) \) code. Moreover, we show how to construct \( FB-(2^s + [(3\alpha - 2) \cdot 2^{s-2}] - 1, s, \left\lceil \alpha \cdot 2^{s-1} \right\rceil) \) codes for all \( 2/3 \leq \alpha \leq 1 \). Another result that can be found in [14] states that
\( FP(s, 2^s) \leq 2^{s+1} - 2 \). In this case, the lower bound is the same, i.e., this result is optimal, see [4]. In this paper we will show that this optimality holds not only for functional PIR codes but also for the more challenging case of functional batch codes, that is, \( FB(s, 2^s) = 2^{s+1} - 2 \). Lastly, we show a non-recursive decoding algorithm for \( B-(2^s, s, k = 2^{s-1}) \) codes. In fact, this construction holds not only for \( k \) single bit requests (with respect to \( k \)-batch codes) but also for \( k \) linear combinations of requests under some constraint that will be explained in the paper. All the results in the paper are achieved using a generator matrix \( G \) of a Hadamard codes [3] of length \( 2^s \) and dimension \( s \), where the matrix’s columns correspond to the servers of the \( FB-(n, s, k) \) code.

There are three reasons we consider the conjectured case \( k = 2^s - 1 \) for functional \( k \)-batch codes. First, this case has already been solved for PIR codes, functional PIR codes, and \( k \)-batch codes. Second, it provides a construction for parallel Random I/O (RIO) codes. Third, this is the case in which every linear combination of the \( s \) information bits is stored in one of the servers.

The rest of the paper is organized as follows. In Section 6.2, we formally define functional \( k \)-batch codes and summarize the main results of the paper. In Section 6.3, we show a construction of \( FB-(2^s + \lfloor (3\alpha - 2) \cdot 2^{s-2} \rfloor - 1, s, \lfloor \alpha \cdot 2^{s-1} \rfloor \) for \( \alpha = 2/3 \). This result is extended for all \( 2/3 \leq \alpha \leq 1 \) in Section 6.4. In Section 6.5, a construction of \( FB-(2^{s+1} - 2, s, 2^s) \) is presented. In Section 6.6, we present our main result, i.e., a construction of \( FB-(2^s - 1, s, \lfloor \alpha \cdot 2^{s-1} \rfloor - s) \) codes. In Section 6.7, a construction of \( B-(2^s - 1, s, 2^{s-1}) \) is presented. Finally, Section 6.8 concludes the paper.

### 6.2 Definitions

For a positive integer \( n \) define \([n] = \{0, 1, \ldots, n-1\}\). All vectors and matrices in the paper are over \( \mathbb{F}_2 \). We follow the definition of functional batch codes as it was first defined in [4].

**Definition 72.** A functional \( k \)-batch code of length \( n \) and dimension \( s \) consists of \( n \) servers and \( s \) information bits \( x_0, x_1, \ldots, x_{s-1} \). Each server stores a nontrivial linear combination of the information bits (which are the coded bits), i.e., for all \( j \in [n] \) exists \( \ell_j \) such that the \( j \)-th server stores a linear combination

\[
y_j = x_{i_0} + x_{i_1} + \cdots + x_{i_{\ell_j-1}},
\]

such that \( i_0, i_1, \ldots, i_{\ell_j-1} \in [s] \). For any request of \( k \) linear bit combinations \( v_0, v_1, \ldots, v_{k-1} \) (not necessarily distinct) of the information bits, there are \( k \) pairwise disjoint subsets \( R_0, R_1, \ldots, R_{k-1} \) of \([n]\) such that the sum of the linear combinations in the related servers of \( R_i \), \( i \in [k] \), is \( v_i \), i.e.,

\[
\sum_{j \in R_i} y_j = v_i.
\]
Each such \( v_i \) will be called a \textit{requested bit} and each such subset \( R_i \) will be called a \textit{recovery set}.

Equivalently, a functional \( k \)-batch code is a linear code with an \( s \times n \) generator matrix

\[ G = [g_0, g_1, \ldots, g_{n-1}] \]

in \( \mathbb{F}_2^{s \times n} \) in which the vector \( g_j \) has ones in positions \( i_0, i_1, \ldots, i_{\ell_j-1} \) if and only if the \( j \)-th server stores the linear combination \( x_{i_0} + x_{i_1} + \cdots + x_{i_{\ell_j-1}} \). Using this matrix representation, a functional \( k \)-batch code is an \( s \times n \) generator matrix \( G \), such that for any \( k \) request vectors \( v_0, v_1, \ldots, v_{k-1} \in \mathbb{F}_2^s \) (not necessarily distinct), there are \( k \) pairwise disjoint subsets of columns in \( G \), denoted by \( R_0, R_1, \ldots, R_{k-1} \), such that the sum of the column vectors whose indices are in \( R_j \) is equal to the \textit{request vector} \( v_j \). The set of all recovery sets \( R_i, i \in [k] \), is called a \textit{solution} for the \( k \) request vectors. The sum of the column vectors whose indices are in \( R_j \) will be called the \textit{recovery sum}.

A functional \( k \)-batch code of length \( n \) and dimension \( s \) over \( \mathbb{F}_2^s \) is denoted by \( FB-(n, s, k) \). Every request of \( k \) vectors will be stored as columns in a matrix \( M \) which is called the \textit{request matrix} or simply the \textit{request}.

In Definition 72 we showed both scalar and vector definitions for \( k \)-batch codes, which are equivalent to each other. While the scalar definition in previous works is the more common one, the vector definition is used mainly in this work because of the unique properties of the generator matrices of these codes.

A \textit{\( k \)-batch code} of length \( n \) and dimension \( s \) over \( \mathbb{F}_2^s \), is denoted by \( B-(n, s, k) \) and is defined similarly to functional \( k \)-batch codes as in Definition 72 except of the fact that each request vector \( v_j \in \mathbb{F}_2^s \) is a unit vector. A functional \textit{\( k \)-PIR code} \[14\] of length \( n \) and dimension \( s \), denoted by \( FP-(n, s, k) \), is a special case of \( FB-(n, s, k) \) in which all the request vectors are identical. We first show some preliminary results on the parameters of \( FB-(n, s, k) \) and \( FP-(n, s, k) \) codes which are relevant to our work. For that, another definition is presented.

**Definition 73.** Denote by \( FB(s, k), B(s, k), FP(s, k) \) the minimum length \( n \) of any \( FB-(n, s, k) \), \( B-(n, s, k) \), \( FP-(n, s, k) \) code, respectively.

Most of the following results on \( FB(s, k), B(s, k) \) and \( FP(s, k) \) can be found in \[14\], while the result in (c) was verified for \( s = 3, 4 \) in \[13\].

**Theorem 74.** For positive integers \( s \) and \( t \), the following properties hold:

(a) \( FP(s, 2^{s-1}) = 2^s - 1 \).

(b) \( FP(st, 2^s) \leq 2t(2^s - 1) \).

(c) For \( s \leq 5 \) it holds that \( FB(s, 2^{s-1}) = 2^s - 1 \).

(d) An \( FB-(2^s - 1, s, 2^{s-2} + 2^{s-4} + \left\lceil \frac{2^{s/2}}{\sqrt{2^s}} \right\rceil) \) code exists.
(e) For a fixed $k$ it holds that
\[
\lim_{{s \to \infty}} \frac{FB(s, k)}{{s}} \geq \frac{k}{\log(k + 1)}.
\]

(f) \( B(s, 2^{s-1}) = 2^s - 1 \) \[13\].

(g) \( B(s, k) = s + \Theta(\sqrt{s}) \) for \( k = 3, 4, 5 \) \[11\].

(h) \( B(s, k) = s + O(\sqrt{s} \log s) \) for \( k > 6 \) \[11\].

Note that the result from Theorem 74(d) improves upon the result of \( FB-(2^s-1, s, 2^{s-2} + 2^{s-4} + 1) \) functional batch codes which was derived from a WOM codes construction by Godlewski \[5\]. This is the best-known result concerning the number of queries when the number of information bits is \( s \) and the number of encoded bits is \( 2^s - 1 \).

The goal of this paper is to improve some of the results summarized in Theorem 74. The result in (c) holds for \( s \leq 5 \), and it was conjectured in \[14\] that it holds for all positive values of \( s \).

**Conjecture 1.** \[14\] For all \( s > 5 \), \( FB(s, 2^{s-1}) = 2^s - 1 \).

The reader can notice the gap between Conjecture 1 and the result in Theorem 74(d). More precisely, \[14\] assures that an \( FB-(2^s-1, s, 2^{s-2} + 2^{s-4} + 1) \) code exists, and the goal is to determine whether an \( FB-(2^s-1, s, 2^{s-1}) \) code exists. This paper takes one more step in establishing this conjecture. Specifically, the best-known value of the number of requested bits \( k \) is improved for the case of \( s \) information bits and \( 2^s - 1 \) encoded bits. The next theorem summarizes the contributions of this paper.

**Theorem 75.** For a positive integer \( s \), the following constructions exist:

(a) A construction of \( FB-(2^s-1, s, \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor ) \) codes.

(b) A construction of
\[
FB-(2^s + \left\lfloor (3\alpha - 2) \cdot 2^{s-2} \right\rfloor - 1, s, \left\lfloor \alpha \cdot 2^{s-1} \right\rfloor )
\]
codes where \( 2/3 \leq \alpha \leq 1 \).

(c) A construction of \( FB-(2^s+1-2, s, 2^s) \) codes.

(d) A construction of \( FB-(2^s-1, s, \left\lfloor \frac{5}{3} \cdot 2^{s-1} \right\rfloor - s) \) codes.

We now explain the improvements of the results of Theorem 75. The construction in Theorem 75(a) improves upon the result from Theorem 74(d), where the supported number of requests increases from \( \frac{1}{2}2^{s-1} + 2^{s-4} + \left\lfloor \frac{2^{s-2}}{\sqrt{24}} \right\rfloor \) to \( \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor \).

Note that by taking \( \alpha = 2/3 \) in the result of Theorem 75(b), we immediately
get the result of \((a)\). However, for simplicity of the proof, we first show the construction for \((a)\) separately, and afterwards, add its extension. The result of Theorem \(75(d)\) is based on the result of Theorem \(75(a)\) and improves it to \(\left\lfloor \frac{5}{6} \cdot 2^{s-1} \right\rfloor - s\) requests. Moreover, according to the second result of Theorem \(74(b)\) if \(t = 1\) then \(FP(s, 2^s) \leq 2^{s+1} - 2\). Based on the result in [4] it holds that \(FP(s, 2^s) \geq 2^{s+1} - 2\). Therefore, \(FP(s, 2^s) = 2^{s+1} - 2\). The construction in Theorem \(75(c)\) extends this result to functional batch codes by showing that \(FB(s, 2^s) \leq 2^{s+1} - 2\), and again, combining the result from [4], it is deduced that \(FB(s, 2^s) = 2^{s+1} - 2\).

A special family of matrices that will be used extensively in the paper are the generator matrices of Hadamard codes [3], as defined next.

**Definition 76.** A matrix \(G = \begin{bmatrix} g_0 & g_1 & \ldots & g_{2^s-1} \end{bmatrix}\) of order \(s \times 2^s\) over \(\mathbb{F}_2\) such that \(\{g_0, g_1, \ldots, g_{2^s-1}\} = \mathbb{F}_2^s\) is called a Hadamard generator matrix and in short \(HG\)-matrix.

We will use \(HG\)-matrices as the generator matrices of the linear codes that will provide the constructions used in establishing Theorem \(75\). More specifically, given a linear code defined by a generator \(HG\)-matrix \(G\) of order \(s \times n\) and a request \(M\) of order \(s \times k\), we will show an algorithm that finds a solution for \(M\). This solution will be obtained by rearranging the columns of \(G\) and thereby generating a new \(HG\)-matrix \(G'\). This solution is obtained by showing all the disjoint recovery sets for the request \(M\), with respect to indices of columns of \(G'\). Although such a solution is obtained with respect to \(G'\) instead of \(G\), it can be easily adjusted to \(G\) by relabeling the indices of the columns. Thus, any \(HG\)-matrix whose column indices are partitioned to recovery sets for \(M\) provides a solution. Note that \(HG\)-matrices store the all-zero column vector. Such a vector will help us to simplify the construction of the algorithm and will be removed at the end of the algorithm.

**Definition 77.** Let \(M = \begin{bmatrix} v_0 & v_1 & \ldots & v_{n/2-1} \end{bmatrix}\) be a request of order \(s \times n/2\), where \(n = 2^s\). The matrix \(M\) has a Hadamard solution if there exists an \(HG\)-matrix \(G = \begin{bmatrix} g_0 & g_1 & \ldots & g_{n-1} \end{bmatrix}\) of order \(s \times n\) such that for all \(i \in [n/2]\),

\[ v_i = g_{2i} + g_{2i+1}. \]

In this case, we say that \(G\) is a Hadamard solution for \(M\).

Next, an example is shown.

**Example 18.** For \(s = 3\), let

\[
G = \begin{pmatrix}
g_0 & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & g_7 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}
\]
be an HG-matrix. Given a request,

\[
M = \begin{pmatrix}
    v_0 & v_1 & v_2 & v_3 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    1 & 1 & 1 & 1
\end{pmatrix}
\]

a Hadamard solution for this request may be

\[
G' = \begin{pmatrix}
    g_0' & g_1' & g_2' & g_3' & g_4' & g_5' & g_6' & g_7' \\
    0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
    0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
    0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}
\]

Lastly, for the convenience of the reader, the relevant notations and terminology that will be used throughout the paper is summarized in Table 6.1.

Table 6.1: Table of Definitions and Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB-(n, s, k)</td>
<td>A func. k-batch code of length n and dimension s</td>
<td>Sec. 6.2</td>
</tr>
<tr>
<td>B-(n, s, k)</td>
<td>A k-batch code of length n and dimension s</td>
<td>Sec. 6.2</td>
</tr>
<tr>
<td>R_i</td>
<td>The i-th recovery set</td>
<td>Sec. 6.2</td>
</tr>
<tr>
<td>(G, B, R)</td>
<td>A triple-set</td>
<td>Def. 78</td>
</tr>
<tr>
<td>(\mathcal{M}(G, B, R))</td>
<td>A triple-matrix of M</td>
<td>Def. 78</td>
</tr>
<tr>
<td>e</td>
<td>A unit vector of length s with 1 at its last index</td>
<td>Sec. 6.3</td>
</tr>
<tr>
<td>M</td>
<td>A request matrix</td>
<td>Sec. 6.3</td>
</tr>
<tr>
<td>(v_i, w_i)</td>
<td>The i-th request/column vector in M, M</td>
<td>Sec. 6.3</td>
</tr>
<tr>
<td>G</td>
<td>An HG-matrix</td>
<td>Sec. 6.3</td>
</tr>
<tr>
<td>(g_i)</td>
<td>A column vector in G representing the i-th server</td>
<td>Sec. 6.3</td>
</tr>
<tr>
<td>(G_x(G))</td>
<td>An x-type graph of G</td>
<td>Def. 81</td>
</tr>
<tr>
<td>(C_x(G))</td>
<td>The partition of simple cycles of (G_x(G))</td>
<td>Def. 81</td>
</tr>
<tr>
<td>(P_x(g_i, g_j))</td>
<td>A simple path between (g_i) and (g_j) in (G_x(G))</td>
<td>Def. 88</td>
</tr>
<tr>
<td>(d_{P_x}(g_i, g_m))</td>
<td>The sub-length from (g_i) to (g_m) in (P_x(g_i, g_j))</td>
<td>Def. 88</td>
</tr>
<tr>
<td>(F_x(g_i, g_j))</td>
<td>A reordering function for a good-path (P_x(g_i, g_j))</td>
<td>Def. 88</td>
</tr>
</tbody>
</table>

6.3 A Construction of \(FB-(2^s - 1, s, \lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor)\) Codes

In this section a construction of \(FB-(2^s - 1, s, \lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor)\) codes is presented. Let the request \(M\) be denoted by

\[
M = [v_0, v_1, \ldots, v_{\lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor}].
\]
Let \( e = (0,0,\ldots,0,1) \in \mathbb{F}_2^s \) be the unit vector with 1 at its last index. The solution for the request \( M \) will be derived by using two algorithms as will be presented in this section. We start with several definitions and tools that will be used in these algorithms.

**Definition 78.** Three sets \( G, B, R \subseteq [2^{s-1}] \) are called a **triple-set** (the good, the bad, and the redundant), and are denoted by \((G, B, R)\), if the following properties hold,

\[
\begin{align*}
    G &\subseteq \left[ \frac{2}{3} \cdot 2^{s-1} \right], \\
    B &\subseteq \left[ \frac{2}{3} \cdot 2^{s-1} \right] \setminus G, \\
    R &\subseteq [2^{s-1}] \setminus (G \cup B \cup \{2^{s-1} - 1\}).
\end{align*}
\]

Given a matrix \( M = [v_0, v_1, \ldots, v_{\lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor - 1}] \) of order \( s \times [\frac{2}{3} \cdot 2^{s-1}] \), the matrix \( M(G, B, R) = [w_0, w_1, \ldots, w_{2^{s-1} - 1}] \) of order \( s \times 2^{s-1} \) is referred as a **triple-matrix of** \( M \) if it holds that

\[
   w_t = \begin{cases} 
   v_t & t \in G \\
   v_t + e & t \in B \\
   e & t \in R
   \end{cases}.
\]

Note that, we did not demand anything about the vector \( w_{2^{s-1} - 1} \), i.e., it can be any binary vector of length \( s \). Furthermore, by Definition 78, the set \( B \) uniquely defines the triple-set \((G, B, R)\). We proceed with the following claim.

**Claim 33.** For any triple-set \((G, B, R)\) if \(|B| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor \) then \(|B| \leq |R|\).

The proof of this claim is shown in Appendix A. As mentioned above, our strategy is to construct two algorithms. We start by describing the first one which is the main algorithm. This algorithm receives as an input the request \( M \) and outputs a set \( B \) and a Hadamard-solution for some triple-matrix \( M(G, B, R) \) of \( M \). Using the matrix \( M(G, B, R) \), it will be shown how to derive the solution for \( M \). This connection is established in the next lemma. For the rest of this section we denote \( n = 2^s \) and for our ease of notations both of them will be used.

**Lemma 79.** If there is a Hadamard solution for \( M(G, B, R) \) such that \(|B| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor \), then there is a solution for \( M = [v_0, v_1, \ldots, v_{\lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor - 1}] \).

**Proof.** Let the \( HG \)-matrix \( G = [g_0, g_1, \ldots, g_{n-1}] \) be a Hadamard solution for \( M(G, B, R) \). Our goal is to form all disjoint recovery sets \( R_t \) for \( t \in G \cup B = [\lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor] \) for \( M \). Since \( G \) is a Hadamard solution for \( M(G, B, R) \), for all \( t \in [2^{s-1}] \), it holds that

\[
   w_t = g_{2t} + g_{2t+1}.
\]
By definition of $M(G, B, R)$

$$w_t = \begin{cases} v_t & t \in G \\ v_t + e & t \in B \\ e & t \in R \end{cases}$$

Thus, if $t \in G$ then

$$v_t = w_t = g_{2t} + g_{2t+1},$$

and each recovery set for $v_t$ is of the form $R_t = \{2t, 2t + 1\}$. If $t \in B$ then

$$v_t + e = w_t = g_{2t} + g_{2t+1},$$

and if $t' \in R$ then

$$e = w_{t'} = g_{2t'} + g_{2t'+1}.$$ 

Therefore, for all $t \in B$ and $t' \in R$,

$$v_t = g_{2t} + g_{2t+1} + g_{2t'} + g_{2t'+1}.$$ 

By Claim 33 since $|B| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor$, it holds that $|B| \leq |R|$. Thus, for all $t \in B$, each recovery set $R_t$ for $v_t$ will have a different $t' \in R$ such that

$$R_t = \{2t, 2t + 1, 2t', 2t' + 1\}.$$ 

Therefore, for all $t \in B$ and $t' \in R$,

$$v_t = g_{2t} + g_{2t+1} + g_{2t'} + g_{2t'+1}.$$ 

By Claim 33, since $|B| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor$, it holds that $|B| \leq |R|$. Thus, for all $t \in B$, each recovery set $R_t$ for $v_t$ will have a different $t' \in R$ such that

$$R_t = \{2t, 2t + 1, 2t', 2t' + 1\}.$$ 

In Lemma 79 it was shown that obtaining $M(G, B, R)$ which holds $|B| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor$ provides a solution for $M$. Therefore, if the first algorithm outputs a set $B$ for which $|B| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor$, then the solution for $M$ is easily derived. Otherwise, the first algorithm outputs a set $B$ such that $|B| > \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor$. In this case, the second algorithm will be used in order to reduce the size of the set $B$ to be at most $\lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor$. For that, more definitions are required, and will be presented in the next section.

### 6.3.1 Graph Definitions

In the two algorithms of the construction, we will use undirected graphs, simple paths, and simple cycles that will be defined next. These graphs will be useful to represent the $HG$-matrix $G$ in some graph representation and to make some swap operations on its columns.

**Definition 80.** An **undirected graph** or simply a graph will be denoted by $G = (V, E)$, where $V = \{u_0, u_1, \ldots, u_{m-1}\}$ is its set of $m$ nodes (vertices) and $E \subseteq \{(u_i, u_j) \mid u_i, u_j \in V\}$ is its edge set. A finite **simple path** of length $\ell$ is a sequence of distinct edges $e_0, e_1, \ldots, e_{\ell-1}$ for which there is a sequence of vertices $u_{i_0}, u_{i_1}, \ldots, u_{i_\ell}$ such that $e_j = \{u_{i_j}, u_{i_{j+1}}\}, j \in [\ell]$. A **simple cycle** is a simple path in which $u_{i_0} = u_{i_\ell}$. The **degree** of a node $u_i$ is the number of edges that are incident to the node, and will be denoted by $\deg(u_i)$. 

181
Given an HG-matrix $G = [g_0, g_1, \ldots, g_{n-1}]$, and a non-zero vector $x \in \mathbb{F}_2^n$, denote the $x$-type graph $G_x(G) = (V, E_x(G))$ of $G$ and $x$ such that $V = \mathbb{F}_2^n$ and a multi-set

$$E_x(G) = \left\{ \{g_i, g_i + x\} \mid i \in [n] \right\} \cup \left\{ \{g_{2t-1}, g_{2t}\} \mid t \in [n/2] \right\}.$$ 

For all $t \in [n/2]$, we say that $g_{2t-1}$ and $g_{2t}$ are a pair. An edge $\{g_{2t-1}, g_{2t}\}$ will be called a pair-type edge and will be denoted by $\{g_{2t-1}, g_{2t}\}_p$. An edge $\{g_i, g_i + x\}$ will be called an $x$-type edge and will be denoted by $\{g_i, g_i + x\}_x$.

Note that for any $g \in V$, it holds that $\deg(g) = 2$. Thus, the graph $G_x(G)$ has a partition of $\ell \geq 1$ disjoint simple cycles, that will be denoted by $C_x(G) = \{C_i\}_{i=0}^{\ell-1}$, where every $C_i$ is denoted by its set of edges.

For the rest of the paper, when the graph $G_x(G)$ is considered, it is assumed that $x \neq 0$. Note that $E_x(G)$ is a multi-set since in case that $g_{2t-1} = g_{2t} + x$, we have two parallel edges $\{g_{2t-1}, g_{2t}\}_p$ and $\{g_{2t-1}, g_{2t}\}_x$ between $g_{2t-1}$ and $g_{2t}$. For the following definitions assume that $G = [g_0, g_1, \ldots, g_{n-1}]$ is an $HG$-matrix of order $s \times n$.

**Definition 82.** Given an $x$-type graph $G_x(G)$ such that $x \in \mathbb{F}_2^n$, let $g_i, g_j$ be two vertices connected by a simple path $P_x(g_i, g_j, G)$ of length $\ell - 1$ in $G_x(G)$ which is denoted by

$$g_i = g_{s_0} - g_{s_1} - \cdots - g_{s_{\ell-1}} = g_j.$$

The path $P_x(g_i, g_j, G)$ will be called a good-path if the edges $\{g_{s_0}, g_{s_1}\}$ and $\{g_{s_{\ell-2}}, g_{s_{\ell-1}}\}$ are both $x$-type edges. For all $g_i$ and $g_m$ on $P_x(g_i, g_j, G)$, denote by $d_{P_x}(g_i, g_m, G)$ the length of the simple sub-path from $g_i$ to $g_m$ on $P_x(g_i, g_j, G)$. This length will be called the sub-length from $g_i$ to $g_m$ in $P_x(g_i, g_j, G)$. When the graph $G$ will be clear from the context we will use the notation $P_x(g_i, g_j)$, $d_{P_x}(g_i, g_m)$ instead of $P_x(g_i, g_j, G)$, $d_{P_x}(g_i, g_m, G)$, respectively.

We next state the following claim.

**Claim 34.** Given a good-path $P_x(g_i, g_j)$ of length $\ell - 1$ in $G_x(G)$

$$g_i = g_{s_0} - g_{s_1} - \cdots - g_{s_{\ell-1}} = g_j,$$

where $x \in \mathbb{F}_2^n$, the following properties hold.
1. The value of \( \ell \) is even.

2. For all \( m \in [\ell/2 - 1] \) the edge \( \{g_{s2m+1}, g_{s2m+2}\}_{P} \) is a pair-type edge.

3. For all \( t \in [\ell/2] \), \( g_{s2t} = g_{s2t+1} + x \).

4. If \( g_i, g_j \) is not a pair, then the pair of \( g_i \) and the pair of \( g_j \) are not in \( P_{x}(g_i, g_j) \).

**Proof.** We prove this claim as follows.

1. Since \( P_{x}(g_i, g_j) \) is a good-path, by definition the edge \( \{g_{s0}, g_{s1}\}_{x} \) is an \( x \)-type edge. We also know that for all \( t \in [\ell] \) it holds that \( \deg(g_{st}) = 2 \). Thus, the edge \( \{g_{s1}, g_{s2}\}_{x} \) is a pair-type edge, the edge \( \{g_{s2}, g_{s3}\}_{x} \) is an \( x \)-type edge, and so on. More formally, for all \( t \in [\ell/2] \) the edge \( \{g_{s2t}, g_{s2t+1}\}_{x} \) is an \( x \)-type edge and for all \( m \in [\ell/2 - 1] \) the edge \( \{g_{s2m+1}, g_{s2m+2}\}_{P} \) is a pair-type edge. Since the last edge \( \{g_{s_{\ell-2}}, g_{s_{\ell-1}}\}_{x} \) is also an \( x \)-type edge, we deduce that \( \ell - 1 \) is odd or equivalently \( \ell \) is even.

2. The proof of this part holds due to a).

3. In a) we proved that for all \( t \in [\ell/2] \) the edge \( \{g_{s_{2t}}, g_{s_{2t+1}}\}_{x} \) is an \( x \)-type edge. Thus, by definition \( g_{s_{2t}} = g_{s_{2t+1}} + x \).

4. Let \( g_m \) be a pair of \( g_i \) and we will prove that \( g_m \notin P_{x}(g_i, g_j) \). Note that \( g_m \neq g_j \) and \( \deg(g_m) = 2 \). Therefore, if \( g_m \in P_{x}(g_i, g_j) \), then \( g_i \) has to appear more than once in \( P_{x}(g_i, g_j) \). This is in contradiction to the fact that \( P_{x}(g_i, g_j) \) is a simple path.

\[ \Box \]

Another useful property on good-paths in \( x \)-type graphs is proved in the next claim.

**Claim 35.** If \( g_i, g_j \) is a pair, then there is a good-path \( P_{x}(g_i, g_j) \) in \( G_{x}(G) \).

**Proof.** We know that all nodes in \( G_{x}(G) \) are of degree 2. Therefore, there is a simple cycle in \( C \in C_{x}(G) \) including the edges \( \{g_i, g_m\}_{x} \) and \( \{g_j, g_p\}_{x} \) for some \( m, p \in [n] \), and the edge \( \{g_i, g_j\}_{p} \). By removing the edge \( \{g_i, g_j\}_{p} \) from \( C \) we get a simple path \( P \) starting with the edge \( \{g_i, g_m\}_{x} \) and ending with the edge \( \{g_j, g_p\}_{x} \). Thus, by definition, \( P \) is a good-path \( P_{x}(g_i, g_j) \).

\[ \Box \]

The next definition will be used for changing the order of the columns in \( G \).

**Definition 83.** Let \( \mathcal{H}_{s} \) be the set of all \( HG \)-matrices of order \( s \times n \). Let \( \mathcal{P}_{s} \subseteq \mathbb{F}_{2}^{s} \times \mathbb{F}_{2}^{s} \) be the set of all couples of column vectors \( g_{m}, g_{p} \) of \( G \) such that there is a good-path \( P_{x}(g_{m}, g_{p}) \). For every two column vectors \( g_{i}, g_{j} \) with a good-path \( P_{x}(g_{i}, g_{j}) \) of length \( \ell - 1 \) in \( G_{x}(G) \)

\[ g_{i} = g_{s_{0}} - g_{s_{1}} - \cdots - g_{s_{\ell-1}} = g_{j}, \]

183
denote the reordering function \( F_x : \mathcal{P}_s \times \mathcal{H}_s \rightarrow \mathcal{H}_s \) that generates an HG-matrix \( F_x(g_i, g_j, G) \) from \( G \) by adding \( x \) to every column \( g_{s_m}, m \in [\ell] \). We will use the notation \( F_x(g_i, g_j) \) for shorthand.

The following claim proves that the function \( F_x \) is well defined.

**Claim 36.** The matrix \( F_x(g_i, g_j) \) is an HG-matrix of order \( s \times n \).

**Proof.** Let \( P_x(g_i, g_j) \) be a good-path of length \( \ell - 1 \) in \( G_x(G) \) denoted by

\[
g_i = g_{s_0} - g_{s_1} - \cdots - g_{s_{\ell-1}} = g_j.
\]

By using the function \( F_x(g_i, g_j) \), the vector \( x \) is added to every column \( g_{s_m}, m \in [\ell] \). In Claim 34(3) it was shown that for all \( t \in [\ell/2] \),

\[
g_{s_{2t}} = g_{s_{2t+1}} + x.
\]

Therefore, adding \( x \) to all the columns \( g_{s_m}, m \in [\ell] \), is equivalent to swapping the column vectors \( g_{s_{2t}}, g_{s_{2t+1}} \) for all \( t \in [\ell/2] \) in \( G \). Since after rearranging the columns of \( G \), it is still an HG-matrix, it is deduced that \( F_x(g_i, g_j) \) is an HG-matrix. \( \Box \)

To better explain these definitions and properties, the following example is presented.

**Example 19.** For \( s = 3 \), let \( G \) be the following HG-matrix

\[
G = \begin{pmatrix}
g_0 & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & g_7 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}.
\]

Let \( x = (1, 0, 1) \). The graph \( G_x(G) \) will be defined as in Figure 6.1. Note that

![Figure 6.1](image)

Figure 6.1: The \( G_x(G) \) graph. The green edges are the \( x \)-type edges and the dashed edges are the pair-type edges.

in this case, the graph \( G_x(G) \) is partitioned into two disjoint cycles. While the path \( g_0 - g_5 \) is a good-path between \( g_0 \) and \( g_5 \), the path

\[
g_0 - g_1 - g_4 - g_5
\]
is not a good-path between \( g_0 \) and \( g_5 \). Note that there is no good-path between \( g_0 \) and \( g_4 \). Let \( P_x(g_0, g_1) \) be the good-path between \( g_0 \) and \( g_1 \).

\[
g_0 - g_5 - g_4 - g_1.
\]

Thus, \( G' = F_x(g_0, g_1) \) is the following HG-matrix

\[
G' = \begin{pmatrix}
g_0 & g_1' & g_2' & g_3' & g_4' & g_5' & g_6' & g_7
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}
\]

with a new graph \( G_x(G') \) as depicted in Figure 6.2.

![Figure 6.2: The graph \( G_x(G') \).](image)

The next lemma shows a very important property that will be used in the construction of the first algorithm. This algorithm will have a routine of \( \lceil \frac{2}{3} \cdot 2^s - 1 \rceil \) iterations. In iteration \( t \leq \lceil \frac{2}{3} \cdot 2^s - 1 \rceil \), we will modify the order of the column vectors of \( G \) such that only the sums \( g_{2t} + g_{2t+1} \) and \( g_{n-2} + g_{n-1} \) will be changed by \( x \in \mathbb{F}_2^s \), and all other sums \( g_{2p} + g_{2p+1} \) where \( p \neq t, n/2 - 1 \) will remain the same. The goal on the \( t \)-th iteration is to get that

\[
g_{2t} + g_{2t+1} = v_t + 1_t e,
\]

where \( 1_t \in \{0, 1\} \) and remember that \( e = (0, 0, \ldots, 0, 1) \in \mathbb{F}_2^s \).

**Lemma 8.4.** Let \( P_x(g_{r_1}, g_{r_2}) \) be a good-path in \( G_x(G) \) where \( x \in \mathbb{F}_2^s \) and \( r_1, r_2 \in [n] \) such that \( g_{r_1}, g_{r_2} \) is not a pair. If \( r_1 \in \{2i, 2i + 1\} \), \( r_2 \in \{2j, 2j + 1\} \) (and note that \( i \neq j \)), then, the HG-matrix

\[
G' = F_x(g_{r_1}, g_{r_2}) = [g_0', g_1', \ldots, g_{n-1}']
\]

satisfies the following equalities

\[
g_{2p}' + g_{2p+1}' = g_{2p} + g_{2p+1} + x \quad \text{for } p \in \{i, j\},
\]

\[
g_{2p}' + g_{2p+1}' = g_{2p} + g_{2p+1} \quad \text{for } p \notin \{i, j\},
\]

where \( p \in [n/2] \).
Proof. We prove this lemma only for \( r_1 = 2i \) and \( r_2 = 2j \) where \( i < j \) while all other cases are proved similarly. Suppose that the good-path \( P_x(g_{2i}, g_{2j}) \) is of length \( \ell - 1 \) and denote it by

\[
g_{2i} = g_{s_0} - g_{s_1} - \cdots - g_{s_{\ell-1}} = g_{2j}.
\]

Let \( S \) be the set \( S = \{s_0, s_1, \ldots, s_{\ell-1}\} \). Let

\[
G' = [g'_0, g'_1, \ldots, g'_{n-1}]
\]

be an \( HG \)-matrix of order \( s \times 2^s \) generated by applying \( F_x(g_{2i}, g_{2j}, G) \). Thus, it is deduced that for all \( m \in [n] \)

\[
\begin{align*}
g'_m &= g_m & \text{if } m \notin S, \\
g'_m &= g_m + x & \text{if } m \in S.
\end{align*}
\]

Since \( P_x(g_{2i}, g_{2j}) \) is as good-path and due to Claim 34, for all \( 1 \leq t \leq \ell/2 - 1 \), it holds that \( \{g_{s_{2t-1}}, g_{s_{2t}}\}_p \) is a pair-type edge. Thus, for all \( 1 \leq t \leq \ell/2 - 1 \)

\[
g'_{s_{2t-1}} + g'_{s_{2t}} = g_{s_{2t-1}} + x + g_{s_{2t}} + x = g_{s_{2t-1}} + g_{s_{2t}}.
\]

Therefore, it is deduced that for all \( p \in [n/2] \setminus \{i, j\} \), it holds that

\[
g'_{2p} + g'_{2p+1} = g_{2p} + g_{2p+1}.
\]

In case that \( p = i \) or \( p = j \), by Claim 34 the columns \( g_{2i+1} \) and \( g_{2j+1} \) are not on the path \( P_x(g_{2i}, g_{2j}) \). Thus, \( g'_{2i+1} = g_{2i+1} \) and \( g'_{2j+1} = g_{2j+1} \). Therefore,

\[
g'_{2p} + g'_{2p+1} = g_{2p} + g_{2p+1} + x.
\]

Before proceeding to the next section, the following \( \text{FindShortPath}(G, x, t, m) \) function is presented. Let \( G \) be an \( HG \)-matrix and \( G_x(G) \) be its graph for some \( x \in \mathbb{F}_2^n \). Let \( \{g_{2t}, g_{2t+1}\}_p \) be a pair-type edge in \( G_x(G) \). Assume that there is another pair-type edge \( \{g_{2m}, g_{2m+1}\}_p \) in \( G_x(G) \) such that \( m > t \). The \( \text{FindShortPath}(G, a, t, m) \) function will be used under the condition that there is a cycle \( C_i \in C_x(G) \) such that both \( \{g_{2t}, g_{2t+1}\}_p \) and \( \{g_{2m}, g_{2m+1}\}_p \) are in \( C_i \).

The \( \text{FindShortPath}(G, x, t, m) \) function is presented since it will be used several times in this paper.

### 6.3.2 The \( \text{FindGoodOrBadRequest}(G, t, B, v) \) function

Let \( G \) be an \( HG \)-matrix, let \( v \in \mathbb{F}_2^n \), let \( t \in [n/2] \), and let \( B \) be a set. Denote \( y = g_{2t} + g_{2t+1} \). In this section we will show the function called \( \text{FindGoodOrBadRequest}(G, t, B, v) \). This function will be used by the first algorithm which will be presented in the next section. The task of this function is to update the
FindShortPath\((G, x, t, m)\)

1: \(P_x \leftarrow \) the good-path \(P_x(g_{2t}, g_{2t+1}, G)\)
2: \(d_1 \leftarrow d_{P_x}(g_{2t+1}, g_{2m})\)
3: \(d_2 \leftarrow d_{P_x}(g_{2t+1}, g_{2m+1})\)
4: if \(d_1 < d_2\) then
5: \(j \leftarrow 2m\)
6: else
7: \(j \leftarrow 2m + 1\)
8: end if
9: Return \(j\)

sum of the pair \(g_{2t} \cdot g_{2t+1}\) to either \(v\) or \(v + y\). It also changes the sum of the last pair \(g_{n-2} \cdot g_{n-1}\), but, this pair is used as a “redundancy pair”, i.e., it is not important what the sum of this pair. Another important thing to mention, is that the algorithm \(\text{FindGoodOrBadRequest}(G, t, B, v)\) do not update the sum of the pairs on indices \(2p\) and \(2p + 1\) for all \(p \neq t\), even though these columns could be reordered. The case \(g_{2t} + g_{2t+1} = v, g_{2t} + g_{2t+1} = v + y\) is called a good, bad case and \(t\) won’t, will be inserted in \(B\), respectively. We now ready to present the function.

An explanation of the \(\text{FindGoodOrBadRequest}(G, t, B, v)\) function is shown in the next example.

**Example 20.** In Fig 6.3 we illustrate three good situations in which Step 20 in the function \(\text{FindGoodOrBadRequest}(G, t, B, v)\) succeeds, and one bad case in which Step 20 in the function \(\text{FindGoodOrBadRequest}(G, t, B, v)\) fails. The solid green line in all figures is a sub-path of the good-path \(P_a\) (which is a path between the nodes \(g_{2t}, g_{2t+1}\) in \(G_a(G)\)). The dashed lines are the pair-type edges. The green dashed line is an edge on \(P_a\). Without loss of generality, it is assumed that the closest node between \(g_{n-2}\) and \(g_{n-1}\) to \(g_{2t+1}\) in \(P_a\) is \(g_{n-2}\). The labels of the edges represent the summation of the vectors of its incident nodes. Each of the three good cases illustrated in (a)-(c) lead to the fact that a pair \(g'_{2t}, g'_{2t+1}\) will be summed up to \(v\) (Step 22). In the bad case illustrated by (d), this pair will be summed up only to \(v + y\) (Steps 26-27).

Denote by \(1 \in \{0, 1\}\) a binary indicator such that \(1 = 1\) if and only if the function \(\text{FindGoodOrBadRequest}(G, t, B, v)\) reaches Step 26. Our next goal is to prove the following important lemma.

**Lemma 85.** The function \(\text{FindGoodOrBadRequest}(G, t, B, v)\) will generate a matrix

\[
G' = [g'_0, g'_1, \ldots, g'_{n-1}]
\]

such that

\[
g'_{2p} + g'_{2p+1} = \begin{cases} 
    g_{2p} + g_{2p+1} & p \neq t, n/2 - 1 \\
    v + y & p = t
\end{cases}
\]
\texttt{FindGoodOrBadRequest}(G, t, B, v) \\
1: \texttt{y} \leftarrow g_{2t} + g_{2t+1} \\
2: \texttt{if } v = \texttt{y} \texttt{ then} \\
3: \quad \texttt{Return } G \texttt{ and } B \\
4: \texttt{end if} \\
5: \texttt{u} \leftarrow g_{2t+1} + g_{n-2} \\
6: \texttt{for } p = 1, 2, 3 \texttt{ do} \\
7: \quad \texttt{if } p = 1 \texttt{ then} \\
8: \quad \quad a \leftarrow v + \texttt{y} \\
9: \quad \texttt{end if} \\
10: \quad \texttt{if } p = 2 \texttt{ then} \\
11: \quad \quad a \leftarrow v + \texttt{y} + \texttt{u} \\
12: \quad \quad \texttt{Swap the columns } g_{2t+1} \texttt{ and } g_{n-2} \texttt{ in } G \\
13: \quad \texttt{end if} \\
14: \quad \texttt{if } p = 3 \texttt{ then} \\
15: \quad \quad a \leftarrow v + \texttt{u} \\
16: \quad \quad \texttt{Swap the columns } g_{2t} \texttt{ and } g_{n-2} \texttt{ in } G \\
17: \quad \texttt{end if} \\
18: \quad P_a \leftarrow \texttt{the good-path } P_a(g_{2t}, g_{2t+1}, G) \\
19: \quad r \leftarrow \{g_{n-2}, g_{n-1}\}_p \\
20: \quad \texttt{if } r \in P_a \texttt{ then} \\
21: \quad \quad j \leftarrow \texttt{FindShortPath}(G, a, t, n/2) \\
22: \quad \quad G' \leftarrow \mathcal{F}_a(g_{2t+1}, g_j) \\
23: \quad \quad \texttt{Return } G' \texttt{ and } B' \\
24: \quad \texttt{end if} \\
25: \texttt{end for} \\
26: \quad G' \leftarrow \mathcal{F}_a(g_{2t}, g_{2t+1}) \\
27: \quad \texttt{Swap the columns } g'_{2t} \texttt{ and } g'_{n-2} \texttt{ of } G' \\
28: \quad B' \leftarrow B \cup \{t\} \\
29: \quad \texttt{Return } G' \texttt{ and } B' \\

\textbf{Proof.} First we show that if the function reaches Step 23, then \\
\[ g_{2t} + g_{2t+1} + a = v. \quad (6.1) \]

We separate the proof for the three cases of \( p \in \{1, 2, 3\} \). To better understand these cases we refer the reader to Fig. 6.3(a)-(c). Remember that by Step 5, \\
\( u = g_{2t+1} + g_{n-2} \).

1. If \( p = 1 \), then \( g_{2t} + g_{2t+1} = \texttt{y} \). By Step 8, \( a = v + \texttt{y} \), and therefore equality (6.1) holds.

188
(a) The $p = 1$ case, $\alpha = v + y$.

(b) The $p = 2$ case, $\alpha = v + y + u$.

(c) The $p = 3$ case, $\alpha = v + u$.

(d) The bad case, $\alpha = v + u$.

Figure 6.3: Explanation of the function $FindGoodOrBadRequest(G, B, t, v)$.

2. If $p = 2$, then by Step 12 after swapping $g_{2t+1}$ and $g_{n-2}$, it is deduced that

$$g_{2t} + g_{2t+1} = y + u.$$  

By Step 11 $\alpha = v + y + u$, which concludes the correctness of equality (6.1).

3. If $p = 3$, then by Step 16 after swapping $g_{2t}$ and $g_{n-2}$, it is deduced that

$$g_{2t} + g_{2t+1} = u.$$  

By Step 15 $\alpha = v + u$, corresponding to (6.1).

Note that by Claim 35 there is always a good-path between $g_{2t}$ and $g_{2t+1}$. Now, suppose that in one of these 3 cases, there is a good-path $P_a(g_{2t}, g_{2t+1})$ in $G_a(G)$ which includes the edge $\{g_{n-2}, g_{n-1}\}_P$ (with respect to Step 20). By executing $FindShortPath(G, \alpha, t, n/2)$ we find the closest node between $g_{n-2}$ and $g_{n-1}$ to $g_{2t+1}$ on the path $P_a$. This node is denoted by $g_j$. Thus, the first and the last edges on the path $P_a(g_{2t+1}, g_j)$ have to be $x$-type edges. By definition, it is deduced that $P_a(g_{2t+1}, g_j)$ is a good-path. According to Step 22 $G' = F_a(g_{2t+1}, g_j)$. By Lemma 34 this step changes the pair summations of only the pairs $g_{2t}, g_{2t+1}$ and $g_{n-2}, g_{n-1}$. More precisely,

$$g'_{2t} + g'_{2t+1} = g_{2t} + g_{2t+1} + \alpha = v$$

and the sum of the pair $g'_{n-2}, g'_{n-1}$ does not matter.

Finally, if the function does not succeed to find any of these good-paths, we will show that it will create a matrix $G'$ such that the pair $g'_{2t}, g'_{2t+1}$ will be almost correct, that is,

$$g'_{2t} + g'_{2t+1} = v + y.$$  

189
This will be done in Steps 26–27. First, the path \( P_a(\mathbf{g}_{2t}, \mathbf{g}_{2t+1}) \) does not include the edge \( \{\mathbf{g}_{n-2}, \mathbf{g}_{n-1}\} \) since Step 20 has failed. Second, the function makes two swaps in Step 12 and Step 16 such that 

\[
\mathbf{g}_{2t} + \mathbf{g}_{2t+1} = \mathbf{u},
\]

and 

\[
\mathbf{g}_{2t+1} + \mathbf{g}_{n-2} = \mathbf{u} + \mathbf{y},
\]

and by Step 15, \( \mathbf{a} = \mathbf{v} + \mathbf{u} \). This is illustrated in Fig. 6.3(d). Thus, according to Step 26, 

\[
\mathbf{G}' = F_a(\mathbf{g}_{2t}, \mathbf{g}_{2t+1})
\]

such that 

\[
\mathbf{g}'_{2t} + \mathbf{g}'_{2t+1} = \mathbf{g}_{2t} + \mathbf{g}_{2t+1} = \mathbf{u},
\]

and the column vectors \( \mathbf{g}_{n-2}, \mathbf{g}_{n-1} \) are not changed. Therefore,

\[
\mathbf{g}'_{2t+1} + \mathbf{g}'_{n-2} = \mathbf{g}_{2t+1} + \mathbf{g}_{n-2} + \mathbf{a} = \mathbf{v} + \mathbf{y}.
\]

By Step 27 in which the columns \( \mathbf{g}'_{2t}, \mathbf{g}'_{n-2} \) are swapped we get

\[
\mathbf{g}'_{2t} + \mathbf{g}'_{2t+1} = \mathbf{v} + \mathbf{y}.
\]

We conclude that the function will generate an \( HG \)-matrix 

\[
\mathbf{G}' = [\mathbf{g}'_0, \mathbf{g}'_1, \ldots, \mathbf{g}'_{n-1}]
\]

such that

\[
\mathbf{g}'_{2p} + \mathbf{g}'_{2p+1} = \begin{cases} 
\mathbf{g}_{2p} + \mathbf{g}_{2p+1} & p \neq t, n/2 - 1 \\
\mathbf{v} + \mathbf{y} & p = t
\end{cases}
\]

where \( \mathbf{1} = 1 \) if and only if the function reached Step 20.

6.3.3 The First Algorithm

We start with the first algorithm which is referred by \( FBSolution(\tau, M) \), where \( M \) is the request and \( \tau \) will be the number of iterations in the algorithm, which is the number of columns in \( M \). We define more variables that will be used in the routine of \( FBSolution(\tau, M) \), and some auxiliary results. The \( \tau \) iterations in the algorithm operate as follows. First, we demand that the initial state of the matrix 

\[
\mathbf{G} = [\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{n-1}]
\]

will satisfy 

\[
\mathbf{g}_{2t} + \mathbf{g}_{2t+1} = \mathbf{e}, \quad t \in [n/2]. \tag{6.2}
\]

The matrix \( \mathbf{G} \) exists due to the following claim.
Claim 37. There is an HG-matrix $G = [g_0, g_1, \ldots, g_{n-1}]$ such that for all $t \in [n/2]$

$$g_{2t} + g_{2t+1} = e.$$ 

Proof. Such an HG-matrix $G$ is constructed by taking an order of its column vectors such that for all $t \in [n/2],

$$g_{2t} = (z_0, z_1, \ldots, z_{s-2}, 0), \quad g_{2t+1} = (z_0, z_1, \ldots, z_{s-2}, 1).$$

The following corollary states that Claim 37 holds for all $x \in F^2_s$ instead of $e$. The proof of this corollary is similar to the one of Claim 37.

Corollary 15. For any $M$ that has one kind of request $v_j$, there is a Hadamard solution for $M$.

Extending Corollary 4 to the cases where there are at most a fixed number of different requests $d$ is an interesting problem by itself, which is out of the scope of this paper. In Section 6.6 we remark that $d = 2$ can be solved by one of our constructions. For the case of $d = 3$ we believe we have a proof, however it is omitted since we found it to be long and cumbersome. Finding a simple solution for this case and in general for arbitrary $d$ is left for future research.

According to Corollary 15, for the rest of this section we assume that $M$ has at least two kinds of requests $v_j$. It is also assumed that $\tau = \lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor$. The HG-matrix at the end of the $t$-th iteration will be denoted by

$$G^{(t+1)} = [g^{(t+1)}_0, g^{(t+1)}_1, \ldots, g^{(t+1)}_{n-1}].$$

Now we are ready to present the $FBSolution(\tau, M)$ algorithm.

Algorithm 1 $FBSolution(\tau, M)$

1: $G^{(0)} \leftarrow G$
2: $B^{(0)} = \emptyset$
3: for $t = 0, \ldots, \tau - 1$ do
4: $G^{(t+1)}, B^{(t+1)} \leftarrow FindGoodOrBadRequest(G^{(t)}, t, B^{(t)}, v_t)$
5: end for
6: Return $G^{(\tau)}$ and $B^{(\tau)}$

At the end of the $FBSolution(\tau, M)$ algorithm we obtained the set $B_1 = B^{(\tau)}$ and the matrix $G^{(\tau)} = [g^{(\tau)}_0, g^{(\tau)}_1, \ldots, g^{(\tau)}_{n-1}]$. By Definition 78, the set $B_1$ uniquely defines the triple-set $(G_1, B_1, R_1)$. Since in our case $y = e$, by Lemma 85, for all $t \in [n/2]$, the matrix $G^{(\tau)}$ satisfies that

$$g^{(\tau)}_{2t} + g^{(\tau)}_{2t+1} = \begin{cases} v_t & t \in G_1 \\ v_t + e & t \in B_1 \\ e & t \in R_1 \end{cases}.$$
Let $M_1 = [w_0, w_1, \ldots, w_{2s-1}]$ be the matrix such that for all $t \in [n/2]$

$$w_t = g_{2t}^{(r)} + g_{2t+1}^{(r)}.$$  

Therefore, it is deduced that $M_1 = M_1(G_1, B_1, R_1)$ is a triple-matrix of $M$. By definition of $M_1(G_1, B_1, R_1)$, the matrix $G^{(r)}$ is its Hadamard solution. If the set $B_1$ satisfies $|B_1| \leq \left\lfloor \frac{1}{3} \cdot 2^s \right\rfloor$ then by Lemma 79 there is a solution for $M$. Otherwise, we will make another reordering on the columns of $G^{(r)}$ in order to obtain a new bad set $B_2$ for which $|B_2| \leq \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor$. This will be done in the next section by showing our second algorithm.

### 6.3.4 The Second Algorithm

From now on we assume that

$$G^{(r)} = G = [g_0, g_1, \ldots, g_{n-1}]$$

and $B = B_1$. Before showing the second we start with the following definition.

**Definition 86.** Let $C_e(G)$ be a partition of simple cycles in $G_e(G)$. A pair of distinct indices $t_1, t_2 \in B$ is called a bad-indices pair in $C_e(G)$ if both edges $\{g_{2t_1}, g_{2t_1+1}\}_p$ and $\{g_{2t_2}, g_{2t_2+1}\}_p$ are in the same simple cycle in $G_e(G)$.

Now we show the algorithm $ClearBadCycles(G, B)$ in which the columns of the $HG$-matrix $G$ are reordered, and the set $B$ will be modified and its size will be decreased.

**Algorithm 2** $ClearBadCycles(G, B)$

1: $C_e(G) \leftarrow$ The partition of simple cycles in $G_e(G)$
2: **while** $\exists t_1, t_2$ a bad-indices pair in $C_e(G)$ **do**
3: \hspace{1em} $j \leftarrow$ FindShortPath($G, e, t_1, t_2$)
4: \hspace{1em} $G \leftarrow F_e(g_{2t_1+1}, g_j)$
5: \hspace{1em} $C_e(G) \leftarrow$ The partition of simple cycles in $G_e(G)$
6: \hspace{1em} Remove $t_1, t_2$ from $B$
7: **end while**
8: Return $G$ and $B$

Let $G_2 = [g_0^*, g_1^*, \ldots, g_{n-1}^*]$ be the $HG$-matrix and $B_2$ be the bad set output of the $ClearBadCycles(G, B)$ algorithm. We remind the reader that $M = [v_0, v_1, \ldots, v_{[n/2]-1}]$. Let $M_2 = [w_0, w_1, \ldots, w_{2s-1}]$ be the matrix such that for all $t \in [n/2]$

$$w_t = g_{2t}^* + g_{2t+1}^*.$$  

Since $B_2$ uniquely defines the triple set $(G_2, B_2, R_2)$, it is deduced that the matrix $M_2$ is a triple-matrix $M_2(G_2, B_2, R_2)$ of $M$. Next, it will be shown that the algorithm $ClearBadCycles(G, B)$ will stop and $|B_2|$ will be bounded from above by $\left\lfloor \frac{1}{3} \cdot 2^s \right\rfloor$ after the execution of the algorithm.
Lemma 87. The algorithm ClearBadCycles(G, B) outputs a set B_2 such that \(|B_2| \leq \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor |.

Proof. According to Step 2, if there is a simple cycle containing a bad-indices pair \(t_1, t_2\) in \(C_e(G)\), the algorithm will enter the routine. Thus, there is a good-path between \(g_{2t_1+1}\) and one of the nodes \(g_{2t_2}, g_{2t_2+1}\) (the closest one between them to \(g_{2t_1+1}\)), and the index of this node is denoted by \(j\) (Step 3). Since \(t_1, t_2 \in B\), before the algorithm reaches Step 4, it holds

\[
g_{2t_1} + g_{2t_1+1} = v_{t_1} + e, \quad g_{2t_2} + g_{2t_2+1} = v_{t_2} + e.
\]

By executing \(\mathcal{F}_e(g_{2t_1+1}, g_j)\), due to Lemma 84, the matrix \(G\) is updated to a matrix \(G'\) such that only the two following pair summations are correctly changed to

\[
g'_{2t_1} + g'_{2t_1+1} = v_{t_1}, \quad g'_{2t_2} + g'_{2t_2+1} = v_{t_2}.
\]

Thus, the indices \(t_1\) and \(t_2\) are removed from \(B\) (Step 3). Therefore, Step 2 will fail when each simple cycle will have at most one \(t \in B\) such that

\[
g_{2t} + g_{2t+1} = v_t + e,
\]

and we will call it a “bad cycle”. Suppose that there are \(p\) bad cycles at the end of the algorithm. We are left with showing that \(p \leq \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor |.

Observe that for all \(t \in \mathcal{R}\), the nodes \(g_{2t}\) and \(g_{2t+1}\) are connected by two parallel edges, and therefore they create cycles of length 2. These cycles are not bad cycles by definition. Also note that if the pair \(g_{n-2}, g_{n-1}\) is in some bad cycle, then the bad pair \(g_{2t}, g_{2t+1}\) in this cycle can be corrected by \(\mathcal{F}_e(g_{2t+1}, g)\), where \(g\) is the closest node between \(g_{n-2}\) and \(g_{n-1}\) to \(g_{2t}\) in this cycle. Therefore, we assume that \(g_{n-2}, g_{n-1}\) is not in any bad cycle. Since there are \(2|\mathcal{R}|\) such columns in \(G\) and together with the pair \(g_{n-2}, g_{n-1}\) and \(|\mathcal{R}| = 2^{s-1} - \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor - 1\), only the first

\[
2 \cdot |\mathcal{B} \cup \mathcal{G}| = 2 \cdot \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor
\]

columns of \(G\) can be partitioned into bad cycles. Our next goal is to prove that the size of each bad cycle is at least 4. Assume to the contrary that there is a bad cycle of length 2. Since we are using the graph \(G_e(G)\), such a simple cycle of two nodes \(g_{2t}, g_{2t+1}, t \in B\), satisfies that

\[
g_{2t} + g_{2t+1} = e.
\]

In that case \(g_{2t} + g_{2t+1} = v_t + e\) since \(v_t\) is non-zero vector, so \(g_{2t} + g_{2t+1} = v_t = e\). According to Step 2 in the function FindGoodOrBadRequest(G, t, B, v), \(t \notin B\), which results with a contradiction. Therefore, indeed all simple cycles are of size at least 4. Thus,

\[
|B| \leq p \leq \left\lfloor \frac{1}{4} \cdot \left(2 \cdot \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor \right) \right\rfloor = \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor.
\]
where the last equality holds since by the nested division \( \left\lfloor \frac{x}{y} \right\rfloor = \left\lfloor \frac{x}{yz} \right\rfloor \) for real \( x, y \) and a positive integer \( z \).

We are finally ready to prove the main result of this section.

**Theorem 88.** An \( FB-(2^s-1, s, \lfloor \frac{3}{2} \cdot 2^{s-1} \rfloor) \) code exists.

**Proof.** By Lemma 87, the algorithm \( ClearBadCycles(G, B) \) outputs the set \( B_2 \) such that its size is at most \( \lfloor \frac{1}{3} \cdot 2^s \rfloor \). The \( HG \)-matrix \( G_2 \) is again a Hadamard solution for a triple-matrix \( M_2(G_2, B_2, R_2) \) of \( M \). Thus, by using Lemma 79, it is deduced that there is a solution for \( M \). After removing the all-zero column vector from \( G \), the proof of this theorem is completed.

### 6.4 A Construction of \( FB-(2^s + \lceil (3\alpha - 2) \cdot 2^{s-2} \rceil - 1, s, \lfloor \alpha \cdot 2^{s-1} \rfloor) \) Codes

In this section we show how to construct \( FB-(2^s + \lceil (3\alpha - 2) \cdot 2^{s-2} \rceil - 1, s, \lfloor \alpha \cdot 2^{s-1} \rfloor) \) codes where \( 2/3 \leq \alpha \leq 1 \). For convenience, throughout this section let \( n = 2^s \) and \( m = 2^s + \lceil (3\alpha - 2) \cdot 2^{s-2} \rceil \). Note that since \( \alpha \geq 2/3 \) it holds that \( m \geq n \).

Let \( e = (0, 0, \ldots, 0, 1) \in \mathbb{F}_2^s \).

The following two definitions extend \( HG \)-matrices from Definition 76 and triple-matrices from Definition 78.

**Definition 89.** A matrix \( G = [g_0, g_1, \ldots, g_{m-1}] \) of order \( s \times m \) over \( \mathbb{F}_2 \) is called an extended-\( HG \)-matrix if the matrix \( H_G = [g_0, g_1, \ldots, g_{n-1}] \) is an \( HG \)-matrix of order \( s \times n \) and for all \( n \leq i \leq m-1 \) it holds \( g_i = e \). The \( HG \)-matrix \( H_G \) will be called the \( H \)-part of \( G \).

**Definition 90.** Three sets \( G, B, R \subseteq [2^{s-1}] \) are called an \( \alpha \)-triple-set, and are denoted by \( \alpha-(G, B, R) \), if the following properties hold

\[
G \subseteq \left[ \left\lfloor \alpha \cdot 2^{s-1} \right\rfloor \right],
B = \left[ \left\lfloor \alpha \cdot 2^{s-1} \right\rfloor \right] \setminus G,
R = [2^{s-1}] \setminus (G \cup B \cup \{2^{s-1} - 1\}).
\]

Given a matrix \( M = [v_0, v_1, \ldots, v_{\left\lfloor \alpha \cdot 2^{s-1} \right\rfloor - 1}] \) of order \( s \times \lfloor \alpha \cdot 2^{s-1} \rfloor \), a matrix \( M(G, B, R) = [w_0, w_1, \ldots, w_{2^{s-1} - 1}] \) of order \( s \times 2^{s-1} \) is referred as an \( \alpha \)-triple-matrix of \( M \) if it holds that

\[
w_t = \begin{cases} 
  v_t & t \in G \\
  v_t + e & t \in B \\
  e & t \in R
\end{cases}
\]
We also know that
\[
\|R\| = \frac{1}{\alpha} \left( \frac{1}{2} \alpha \cdot 2^{s-1} \right)
\]
will similarly lead to a solution for \(M\). Then it is possible to construct a solution for \(M\) will be done based on the property that for all \(t \in R\), the size of \(R\) will not be bigger than the size of \(B\) for \(\alpha > 2/3\). Thus, in this case, not all bad summation can be corrected. For that, we define the set \(\mathcal{N}\) that is also used to correct the summations \(v_t + e\) to \(v_t\), where \(t \in B\). This will be done based on the property that for all \(t \in \mathcal{N}\) it holds that \(g_t = e\). In case that \(\alpha < 1\), together with \(R\) and \(\mathcal{N}\), the last pair \(g_{n-2}, g_{n-1}\) will be used for the correction of these summations. Thus, if the inequality \(|B| \leq |R| + |\mathcal{N}| + 1\) holds, then it is possible to construct a solution for \(M\). In case that \(\alpha = 1\), we obtain \(|R| = 0\). In this case, we will show how to get the inequality \(|B| \leq |\mathcal{N}|\), which will similarly lead to a solution for \(M\). Even though the last pair \(g_{n-2}, g_{n-1}\) has an arbitrary summation, it will still be shown how to obtain the request \(v_{n/2-1}\) from this pair. Therefore, our first goal is to show a condition which assures that either \(|B| \leq |R| + |\mathcal{N}| + 1\) or \(|B| \leq |\mathcal{N}|\). This is done in Claim 38

**Claim 38.** Let \((G, B, R)\) be an \(\alpha\)-triple-set where \(|B| \leq \left\lfloor \frac{1}{2} \alpha \cdot 2^{s-1} \right\rfloor\). If \(2/3 \leq \alpha < 1\), then \(|B| \leq |R| + |\mathcal{N}| + 1\), and if \(\alpha = 1\) then \(|B| \leq |\mathcal{N}|\).

**Proof.** Let \(2/3 \leq \alpha < 1\). According to the definition of \(\alpha\)-(\(G, B, R\)), since \(G \cup B = \left(\left\lfloor \alpha \cdot 2^{s-1} \right\rfloor\right)\) it holds that

\[
|R| = \left| \left(2^{s-1}\right) \right| \setminus \left( G \cup B \cup \left\{2^{s-1} - 1\right\} \right) \\
= 2^{s-1} - \left\lfloor \alpha \cdot 2^{s-1} \right\rfloor - 1.
\]

We also know that \(|\mathcal{N}| = \left(3\alpha - 2\right) \cdot 2^{s-2}\). Thus, in order to prove that \(|B| \leq |R| + |\mathcal{N}| + 1\), since \(|B| \leq \left\lfloor \frac{1}{2} \alpha \cdot 2^{s-1} \right\rfloor\), it is enough to prove inequality (a) in

\[
|\mathcal{R}| + |\mathcal{N}| + 1 = 2^{s-1} - \left\lfloor \alpha \cdot 2^{s-1} \right\rfloor + \left(3\alpha - 2\right) \cdot 2^{s-2} \\
\geq \left\lfloor \frac{1}{2} \alpha \cdot 2^{s-1} \right\rfloor + |B|.
\]
Inequality (a) is equivalent to
\[ 2^{s-1} \geq \left\lceil 2\alpha \cdot 2^{s-2} \right\rceil + \left\lceil \alpha \cdot 2^{s-2} \right\rceil - \left\lceil (3\alpha - 2) \cdot 2^{s-2} \right\rceil, \]
which holds since
\[
\begin{align*}
\left\lceil 2\alpha \cdot 2^{s-2} \right\rceil + \left\lceil \alpha \cdot 2^{s-2} \right\rceil - \left\lceil (3\alpha - 2) \cdot 2^{s-2} \right\rceil &\leq 2\alpha \cdot 2^{s-2} + \alpha \cdot 2^{s-2} - (3\alpha - 2) \cdot 2^{s-2} \\
&= 2^{s-2}(2\alpha + \alpha - (3\alpha - 2)) = 2 \cdot 2^{s-2} = 2^{s-1}.
\end{align*}
\]

Now if \( \alpha = 1 \), then \(|\mathcal{B}| \leq \left\lceil \frac{1}{2} \cdot 2^{s-1} \right\rceil = 2^{s-2} \). By the definition of \( \mathcal{R} \), it holds that \(|\mathcal{R}| = 0 \) and by the definition of \( \mathcal{N} \) it holds that \(|\mathcal{N}| = 2^{s-2} \). Therefore, \(|\mathcal{B}| \leq 2^{s-2} = |\mathcal{N}|. \)

\[ \square \]

Let \( M = [v_0, v_1, \ldots, v_{\left\lceil \alpha \cdot 2^{s-1} \right\rceil - 1}] \) be a request of order \( s \times |\alpha \cdot 2^{s-1}| \). Our goal is to construct an extended-\( HG \)-matrix of order \( s \times m \) which will provide a solution for \( M \). For that, the \( \alpha\text{-FBSolution}(M) \) algorithm is presented. In this algorithm, the matrix \( H \) is represented by \( \hat{H} = [g_0, g_1, \ldots, g_{n-1}] \).

**Algorithm 3 \( \alpha\text{-FBSolution}(M) \)**

1. if \( \alpha < 1 \) then
2. \( \tau \leftarrow \left\lceil \alpha \cdot 2^{s-1} \right\rceil \)
3. else if \( \alpha = 1 \) then
4. \( \tau \leftarrow 2^{s-1} - 1 \)
5. end if
6. \( H, \mathcal{B} \leftarrow \text{FBSolution}(\tau, M) \)
7. \( H, \mathcal{B} \leftarrow \text{ClearBadCycles}(H, \mathcal{B}) \)
8. if \( \alpha < 1 \) and \( g_{n-2} \neq e \) then
9. \( H \leftarrow F_{g_{n-2}+e}(g_{n-2}, g_{n-1}) \)
10. end if
11. if \( \alpha = 1 \) and \( g_{n-2} \neq v_{n/2-1} \) then
12. \( H \leftarrow F_{g_{n-2}+v_{n/2-1}}(g_{n-2}, g_{n-1}) \)
13. end if
14. Return \( H \) and \( \mathcal{B} \)

Denote by \( G = [g_0, g_1, \ldots, g_{m-1}] \) an extended-\( HG \)-matrix of order \( s \times m \) such that the output matrix \( H \) from the \( \alpha\text{-FBSolution}(M) \) algorithm is its \( H \)-part, i.e., \( H_G = H \). Note that Steps \[3, 7\] define the set \( \mathcal{B} \). This set is obtained using a similar technique to the one from Section \[3.3\] except to the fact that here \( 2/3 \leq \alpha \leq 1 \), while in Section \[3.3\] \( \alpha = 2/3 \). It is important to note that the size of \( \mathcal{B} \) is bounded due to the execution of the \( \text{ClearBadCycles}(H, \mathcal{B}) \) algorithm (Step \[7\]). Therefore, we only state the following lemma since its proof is very similar to the one that was shown in Lemma \[87\].

**Lemma 91.** The \( \alpha\text{-FBSolution}(M) \) algorithm outputs a set \( \mathcal{B} \) such that \(|\mathcal{B}| \leq \left\lceil \frac{1}{2} \alpha \cdot 2^{s-1} \right\rceil \).
We will use Lemma 91 while proving the main theorem of this section.

**Theorem 92.** For any \(2/3 \leq \alpha \leq 1\), a functional batch code

\[
FB-(2^s + [(3\alpha - 2) \cdot 2^{s-2}] - 1, s, [\alpha \cdot 2^{s-1}])
\]

exists.

**Proof.** After finishing the \(\alpha\)-FB\(S\)(\(M\)) algorithm, we obtain an \(HG\)-matrix \(H\) which is the \(H\)-part of the extended-\(HG\)-matrix \(G = [g_0, g_1, \ldots, g_m]\). Remember that by the definition of \(G\) and \(N\), for all \(t \in N\), it holds that \(g_t = e\). In Step 6 we invoke the algorithm \(FB\(S\)(\(\tau,M\)). Therefore, for all \(t < \tau\), there exists \(1_t \in \{0,1\}\) such that

\[
g_{2t} + g_{2t+1} = v_t + 1_t e.
\]

Let \(\alpha-(G, B, R)\) be an \(\alpha\)-triple-set that is uniquely defined by \(B\) according to Definition 90. Clearly, for all \(t \in G\), the recovery set \(R_t\) is \(R_t = \{2t, 2t + 1\}\). By Lemma 91 it holds that \(|B| \leq |R| + |N| + 1\). We separate this proof for two cases.

**Case 1:** Assume that \(\alpha < 1\). Due to Lemma 91 and Claim 38 if \(\alpha < 1\), it is deduced that \(|B| \leq |R| + |N| + 1\). Let \(t\) be the maximum number in \(B\) and let \(B' = B \setminus \{t\}\). Thus, \(|B'| \leq |R| + |N|\). Therefore, for all \(t \in B'\), \(R_t\) will have a different \(t'\) such that \(R_t\) equals to either \(\{2t, 2t + 1, 2t', 2t' + 1\}\) where \(t' \in R\), or \(\{2t, 2t + 1, t'\}\) where \(t' \in N\). Thus, we showed the recovery sets for all requests except of \(v_t\). Remember that \(g_{2t} + g_{2t+1} = v_t + e\), and note that if \(g_{n-2} = e\), this case is finished. Otherwise, \(g_{n-2} \neq e\). This is handled by Steps 8–9 as follows. By Claim 35 since \(g_{n-2}\) and \(g_{n-1}\) is a pair, we know that there is a good-path \(P_x(g_{n-2}, g_{n-1})\) in an \(x\)-type graph \(G_x(G)\) for all \(x \in \mathbb{F}_2^\times\). Thus, if \(g_{n-2} \neq e\), by taking \(x = g_{n-2} + e\), the algorithm can use the reordering function \(\mathcal{F}_{g_{n-2}+e}(g_{n-2}, g_{n-1})\), as it is done in Step 9. By Lemma 84, we obtain two new column vectors \(g'_{n-2}\) and \(g'_{n-1}\) such that

\[
\begin{align*}
g'_{n-2} &= g_{n-2} + g_{n-2} + e = e, \\
g'_{n-1} &= g_{n-1} + g_{n-2} + e,
\end{align*}
\]

without changing the summations of all other pairs on this path. Therefore, the recovery set for \(v_t\) will be \(R_t = \{2t, 2t + 1, n - 2\}\), which concludes this case.

**Case 2:** Assume that \(\alpha = 1\). Due to Lemma 91 and Claim 38 if \(\alpha = 1\) then \(|B| \leq |N|\). Thus, similarly to Case 1, for all \(t \in B\), the recovery sets \(R_t\) can be obtained. However, we do not have a recovery set for \(v_{n/2-1}\) since the sum of the pair \(g_{n-2}, g_{n-1}\) is arbitrary. If \(g_{n-2} = v_{n/2-1}\), then \(R_{n/2-1} = \{n - 2\}\). Otherwise, as in Case 1, by Step 12 it is deduced that \(g'_{n-2} = v_{n/2-1}\). Again \(R_{n/2-1} = \{n - 2\}\), which concludes this case.

In both cases, after removing the all-zero column from \(G\), we conclude the proof. \(\square\)
6.5 A Construction of \( FB-(2^{s+1} - 2, s, 2^s) \) Codes

In this section, a construction for \( FB-(2^{s+1} - 2, s, 2^s) \) codes will be shown by using the algorithm \( FBSolution(\tau, M) \). Throughout this section let \( n = 2^{s+1} \) and let \( e = (0,0,\ldots,0,1) \in \mathbb{F}_2^{s+1} \). We start with the following definition.

**Definition 93.** A matrix \( G = [g_0, g_1, \ldots, g_{2^{s+1}-1}] \) of order \( s \times 2^{s+1} \) over \( \mathbb{F}_2 \) such that each vector of \( \mathbb{F}_2^s \) appears as a column vector in \( G \) exactly twice, is called a double-\( HG \)-matrix.

Note that by removing the last row from any \( HG \)-matrix of order \((s+1) \times n\), we get a double-\( HG \)-matrix of order \( s \times n \). Also, note that each double-\( HG \)-matrix has exactly two all-zero columns. These columns will be removed at the end of the procedure, obtaining only \( 2^{s+1} - 2 \) column vectors. Next, the definition of a Hadamard solution is extended with respect to Definition 77.

**Definition 94.** Let \( M = [v_0, v_1, \ldots, v_{2^s-1}] \) be a request of order \( s \times 2^s \). The matrix \( M \) has a Hadamard solution if there exists a double-\( HG \)-matrix \( G = [g_0, g_1, \ldots, g_{2^{s+1}-1}] \) of order \( s \times n \) such that for all \( t \in [2^s - 1] \),

\[
v_t = g_{2t} + g_{2t+1},
\]

and for \( t = n/2 - 1 \) either \( v_t = g_{n-2} + g_{n-1} \), or \( v_t = g_{n-2} \), or \( v_t = g_{n-1} \).

Let \( M = [v_0, v_1, \ldots, v_{2^s-1}] \) be a request of order \( s \times 2^s \). Our goal is to construct a double-\( HG \)-matrix of order \( s \times 2^{s+1} \) which will provide a Hadamard solution for \( M \). Let \( M = [w_0, w_1, \ldots, w_{2^s-1}] \) be a new matrix of order \((s+1) \times 2^s\) generated by adding the all-zero row to \( M \). Let \( \tau = n/2 - 1 \) and \( 0_\ell \) be the all-zero vector of length \( \ell \). We now show the algorithm \( OptFBSolution(M) \), which receives as an input the matrix \( M \) and outputs a double-\( HG \)-matrix \( G \) that will be a solution for \( M \). As mentioned in the Introduction the returned solution is optimal.

**Algorithm 4 OptFBSolution(M)**

1. \( \tau \leftarrow n/2 - 1 \)
2. \( G \leftarrow FBSolution(\tau, M) \)
3. \( y \leftarrow \sum_{i=0}^{2^s-1} w_i \)
4. if \( y \neq 0_{s+1} \) and \( y \neq g_{n-2} \) then
5. \( G \leftarrow Fg_{n-2} + y(g_{n-2}; g_{n-1}) \)
6. end if
7. Remove the last row from \( G \)

The following lemma proves the correctness of Algorithm \( OptFBSolution(M) \).

**Lemma 95.** The algorithm \( OptFBSolution(M) \) outputs a double-\( HG \)-matrix \( G' \) which is a Hadamard solution for \( M \).
Proof. According to Step 2, the algorithm \( FBSolution(\tau, \mathcal{M}) \) is used with \( \tau = \frac{n}{2} - 1 \). Thus, by Lemma 85 we obtain an \( HG \)-matrix \( G = [g_0, g_1, \ldots, g_{n-1}] \) such that for all \( t \in [n/2 - 1] \)

\[
g_{2t} + g_{2t+1} = w_t + 1_t e,
\]

where \( 1_t \in \{0, 1\} \). Let \( G' = [g'_0, g'_1, \ldots, g'_{n-1}] \) be a double-\( HG \)-matrix of order \( s \times n \) generated by removing the last row from \( G \) according to Step 7. Since \( \mathcal{M} = [w_0, w_1, \ldots, w_{2^s-1}] \) is generated by adding the all-zero row to \( M \), by removing the last row from \( G \) before Step 3 for all \( t \in [n/2 - 1] \) we could obtain \( G' \) such that

\[
g'_{2t} + g'_{2t+1} = v_t.
\]

However, \( G' \) would provide a solution for \( M \) except for the last request \( v_{n/2-1} \). We handle the last request using Steps 3–5 that will be explained as follows.

Assume that \( \sum_{t=0}^{2^s-1} w_t = y \) and note that

\[
\sum_{i=0}^{2^{s+1}-1} g_i = 0_{s+1}.
\]

Denote

\[
1_{n/2-1} = \sum_{t=0}^{2^s-2} 1_t (\mod 2).
\]

Thus, it is deduced that

\[
g_{n-2} + g_{n-1} \overset{(a)}{=} \sum_{i=0}^{2^{s+1}-3} g_i \overset{(b)}{=} \sum_{t=0}^{2^s-2} (w_t + 1_t e) \overset{(c)}{=} w_{n/2-1} + y + 1_{n/2-1} e.
\]

Equality \( (a) \) holds due to \( (6.5) \), equality \( (b) \) holds according to \( (6.3) \), and equality \( (c) \) holds by the definition of \( y \) and by \( (6.6) \). Now if \( y = 0_{s+1} \), according to Step 6, after removing the last row from \( G \), we get \( G' \) such that equation \( (6.4) \) holds also for \( t = n/2 - 1 \), that is,

\[
g'_{n-2} + g'_{n-1} = v_{n/2-1}.
\]

Clearly, in this case \( G' \) is a Hadamard solution for \( M \). Otherwise, if \( y \neq 0_{s+1} \) then the algorithm enters the if condition in Step 4. By Claim 35, since \( g_{n-2} \) and \( g_{n-1} \) is a pair, we know that there is a good-path \( P_x(g_{n-2}, g_{n-1}) \) in an \( x \)-type graph \( G_x(G) \) for all \( x \in \mathbb{F}^{s+1}_2 \). Thus, by taking \( x = g_{n-2} + y \), the algorithm will
execute the reordering function $F_{g_{n-2} + y}(g_{n-2}, g_{n-1})$ (Step 5). By Lemma 84, we obtain two new column vectors $g'_{n-2}$ and $g'_{n-1}$ such that

$$
\begin{align*}
g'_{n-2} &= g_{n-2} + g_{n-2} + y = y, \\
g'_{n-1} &= g_{n-1} + g_{n-2} + y = w_{n/2-1} + \mathbb{I}_{n/2-1}e,
\end{align*}
$$

without changing the summation of all other pairs on this path. Again, by removing the last row from $G$, we obtain $G'$ such that

$$g'_{2t} + g'_{2t+1} = v_t, \quad t \in [n/2 - 1]$$

and $g'_{n-1} = v_{n/2-1}$. Thus, all the recovery sets $R_t, t \in [n/2 - 1]$ are of the form $R_t = \{2t, 2t + 1\}$, and the last recovery set will be $R_{n/2-1} = \{n - 1\}$, which concludes this case. In both cases, $G'$ is a double-$HG$-matrix with two all-zero columns that will be removed to provide an $FB-(2^{s+1} - 2, s, 2^s)$ code.

For the rest of the paper, we only state that it is possible to obtain the last recovery set from the redundancy columns $g_{n-2}$ and $g_{n-1}$, as it was shown in the proof of Lemma 95. From the result of Lemma 95 we deduce the main theorem of this section.

**Theorem 96.** An $FB-(2^{s+1} - 2, s, 2^s)$ code exists.

### 6.6 A Construction of $FB-(2^s - 1, s, \lfloor \frac{5}{6} \cdot 2^{s-1} \rfloor - s)$ Codes

In this section we show how to improve our main result, i.e., we show a construction of $FB-(2^s - 1, s, \lfloor \frac{5}{6} \cdot 2^{s-1} \rfloor - s)$ codes. Let $M$ be a request denoted by

$$M = [v_0, v_1, \ldots, v_{\lfloor \frac{5}{6} \cdot 2^{s-1} \rfloor - s}].$$

Remember that $e = (0, 0, \ldots, 0, 1) \in \mathbb{F}_2^s$, and $n = 2^s$. The initial state of the matrix

$$G = [g_0, g_1, \ldots, g_{n-1}]$$

will satisfy

$$g_{2t} + g_{2t+1} = e, \quad t \in [n/2].$$

Remember that for all $x \in \mathbb{F}_2^s$, the graph $G_x(G)$ has a partition of $\ell \geq 1$ disjoint simple cycles, that will be denoted by $C_x(G) = \{C_i\}_{i=0}^{\ell-1}$ (Definition 81). Fix $\tau \in [n/2]$. The first ingredient in the solution of $FB-(2^s - 1, s, \lfloor \frac{5}{6} \cdot 2^{s-1} \rfloor - s)$ codes will be presented in algorithm $FBSolution2(\tau, M)$, which is presented as Algorithm 5.

In the internal routine starting on Step 3 on its $t$-th iteration, the algorithm will try to find two column vectors $g^{(t)}_b$ and $g^{(t)}_h$, such that $h, p \geq 2t$, and a request $v_m$, where $m \geq t$, such that the sum of $g^{(t)}_b$ and $g^{(t)}_h$ could be updated
Algorithm 5  \textit{FBSolution2}(\tau, M)

1: \( G^{(0)} \leftarrow G \)
2: \textbf{for} \( t = 0, \ldots, \tau - 1 \) \textbf{do}
3: \hspace{1em} \textbf{for} all \( 2t \leq p, h \leq n - 1 \) and \( t \leq m \leq \tau - 1 \) \textbf{do}
4: \hspace{2em} Swap \( g^{(t)}_p \) and \( g^{(t)}_{2t} \)
5: \hspace{2em} Swap \( g^{(t)}_h \) and \( g^{(t)}_{2t+1} \)
6: \hspace{2em} \textbf{if} \( g^{(t)}_{2t} + g^{(t)}_{2t+1} = v_m \) \textbf{then}
7: \hspace{3em} Swap between \( v_t \) and \( v_m \) in \( M \)
8: \hspace{2em} \textbf{Go to Step 2} \\
9: \hspace{2em} \textbf{end if}
10: \hspace{2em} \( a_m \leftarrow v_m + g^{(t)}_{2t} + g^{(t)}_{2t+1} \)
11: \hspace{2em} Let \( C_i \in Ca_m(G) \) be a cycle s.t. \( \{g^{(t)}_{2t}, g^{(t)}_{2t+1}\}_p \in C_i \)
12: \hspace{2em} \textbf{if} \( \{g^{(t)}_{2t}, g^{(t)}_{2t+1}\}_p \in C_i \) s.t. \( \ell > t \) \textbf{then}
13: \hspace{3em} Swap between \( v_t \) and \( v_m \) in \( M \)
14: \hspace{2em} \textbf{Go to Step 19} \\
15: \hspace{2em} \textbf{end if}
16: \hspace{2em} \textbf{end for}
17: \hspace{2em} \( G^{(t)}, M \leftarrow \text{BadCaseCorrection}(G^{(t)}, a_m, M) \)
18: \hspace{2em} \textbf{Go to Step 3} \\
19: \hspace{2em} \( j \leftarrow \text{FindShortPath}(G^{(t)}, a_m, t, \ell) \)
20: \hspace{2em} \( G^{(t+1)} \leftarrow \mathcal{F}_{a_j}(g^{(t)}_{2t+1}, g^{(t)}_j) \)
21: \hspace{2em} \textbf{end for}
22: \textbf{Return} \( G^{(\tau)} \)

We notice that the \textit{FBSolution3}(G, \tau, M) algorithm will be invoked only if

\[
\left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor \leq \left\lfloor \frac{5}{6} \cdot 2^{s-1} \right\rfloor - s,
\]  \hfill (6.7)

which holds for \( s \geq 7 \).
6.6.1 The $\tau \leq n/4$ Case

Before proving the correctness of the $FBSolution2(\tau, M)$ algorithm, we start with an important definition.

**Definition 97.** On the $t$-th iteration, a path $P_x$ between $g^{(t)}_p, p \geq 2t$ and $g^{(t)}_h, h \geq 2t$ will be called a **short-path** in $G_x(G^{(t)})$, if it is a good-path, and all the pair-type edges $\{g^{(t)}_{2t'}, g^{(t)}_{2t'+1}\}_P$ on $P_x$ satisfy $t' < t$. The short-path $P_{g^{(t)}_p, g^{(t)}_h}$ is called a **trivial short-path**.

Our first goal is to show that every $g^{(t)}_p, p \geq 2t$ has $n$ different short-paths ending on $n$ columns $g^{(t)}_h, h \geq 2t$.

**Claim 39.** Fix some $g^{(t)}_p$ such that $p \geq 2t$. Then, for each $G_x(G^{(t)})$, there exists $g^{(t)}_h, h \geq 2t$, such that there is a short-path between $g^{(t)}_p$ and $g^{(t)}_h$.

**Proof.** Given $g^{(t)}_p$ such that $p \geq 2t$, its pair $g^{(t)}_{p'}$ also satisfies $p' \geq 2t$. In Claim 35, we proved that for all $x \in \mathbb{F}_2^n$ every pair $\{g^{(t)}_{2m}, g^{(t)}_{2m+1}\}_p$ has a good-path $P_x(g^{(t)}_{2m}, g^{(t)}_{2m+1})$ in $G_x(G)$. Therefore, by Claim 35, for all $x \in \mathbb{F}_2^n$, there is a good-path $P_x(g^{(t)}_p, g^{(t)}_{p'})$ in $G_x(G^{(t)})$. If for all the edges $\{g^{(t)}_{2m}, g^{(t)}_{2m+1}\}_p$ on $P_x(g^{(t)}_p, g^{(t)}_{p'})$ it holds that $m < t$, then $P_x(g^{(t)}_p, g^{(t)}_{p'})$ is a short-path. Otherwise, there exists an edge $\{g^{(t)}_{2m}, g^{(t)}_{2m+1}\}_p$ on $P_x(g^{(t)}_p, g^{(t)}_{p'})$, such that $m \geq t$, and without loss of generality, we assume that this edge is the closest one to $g^{(t)}_p$ on $P_x(g^{(t)}_p, g^{(t)}_{p'})$. Let $h \in \{2m, 2m+1\}$ such that the column $g^{(t)}_h$ is the closest node between $g^{(t)}_{2m}$ and $g^{(t)}_{2m+1}$ to $g^{(t)}_p$ on $P_x(g^{(t)}_p, g^{(t)}_{p'})$. Therefore $h \geq 2t$ and this sub-path is a short-path by definition. 

Next, we proceed to prove the correctness of the $FBSolution2(\tau, M)$ algorithm. On Step 10, the algorithm will execute $a_m = v_m + g^{(t)}_p + g^{(t)}_h$. If $g^{(t)}_p$ and $g^{(t)}_h$ have a non-trivial short-path between them in $G_a(G^{(t)})$, then our technique cannot update the sum of $g^{(t)}_p$ and $g^{(t)}_h$ to be equal to $v_m$ without changing the sum of a pair $\{g^{(t)}_{2t'}, g^{(t)}_{2t'+1}\}_p$ for some $t' < t$. So our goal is to find columns $g^{(t)}_p$ and $g^{(t)}_h$ such that there are no (non-trivial) short-paths between them. We state in the following claim that reordering the columns $g^{(t)}_p$ for $p \geq 2t$ of $G^{(t)}$ does not affect their short-paths.

**Claim 40.** The columns $g^{(t)}_p$ and $g^{(t)}_h$ (before Steps 4-5) have no (non-trivial) short-paths between them, if and only if the columns $g^{(t)}_{2t}$ and $g^{(t)}_{2t+1}$ (after Steps 4-5) have no (non-trivial) short-paths between them.

**Proof.** We will prove only the first direction, while the second is proved similarly. In Claim 39 we proved that for all $x \in \mathbb{F}_2^n$, there exists $g^{(t)}_{p'}, p' \geq 2t$ such that
there is a short-path between \( g_p^{(t)} \) and \( g_p^{(t')} \). In this claim we assume that every such \( p' \) satisfies \( p' \neq h \). By definition of the short-path, the edge \( \{g_2^{(t)}, g_2^{(t+1)}_p\} \) is not on any of these short-paths. Therefore, for all \( x \in F_2^s \), the execution of Step 4 will not affect these short-paths. Similarly, for all \( x \in F_2^s \) the short-paths between \( g_h^{(t)} \) and \( g_h^{(t)} \) will not be affected by the execution of Step 5 \( (h' \) is defined similarly to \( h \)). Thus, the columns \( g_2^{(t)} \) and \( g_2^{(t+1)} \) (after Steps 4–5) will not have any (non-trivial) short-path between them.

\[ \square \]

Using Claim 40 we can make columns \( g_p^{(t)} \) and \( g_h^{(t)} \) to be a pair. This is done by Steps 4 and 5, i.e., this pair is now \( \{g_2^{(t)}, g_2^{(t+1)}_p\} \). In the next lemma we will use the properties of short-paths to prove the correctness of the algorithm.

**Lemma 98.** On the \( t \)-th iteration, if there are no (non-trivial) short-paths between \( g_2^{(t)} \) and \( g_2^{(t+1)} \), then by the end of this iteration it holds that

\[ g_2^{(t+1)} + g_2^{(t+1)} = v_m, \]

and all the pair sums \( g_2^{(t+1)} + g_2^{(t+1)} \) such that \( t' < t \) will be unchanged.

**Proof.** If \( g_2^{(t)} + g_2^{(t+1)} = v_m \), then due to Step 6 this lemma is correct. Otherwise, we know that there are no (non-trivial) short-paths between \( g_2^{(t)} \) and \( g_2^{(t+1)} \). Therefore, Step 12 will succeed to find \( \{g_2^{(t)}, g_2^{(t+1)}_p\} \in C_i \) such that \( \ell > t \). Thus, there is a good-path between \( g_2^{(t)} \) and one of the nodes \( g_2^{(t)}, g_2^{(t+1)} \) (the closest one between them to \( g_2^{(t+1)} \)), and the index of this node is denoted by \( j \) (Step 19). By executing \( F_{a_m}(g_2^{(t+1)}, g_j) \), the matrix \( G^{(t)} \) is updated to a matrix \( G^{(t+1)} \) such that only the two following pair summations are correctly changed to

\[ g_2^{(t+1)} + g_2^{(t+1)} = g_2^{(t)} + g_2^{(t+1)} + a_m = v_m \]

\[ g_2^{(t+1)} + g_2^{(t+1)} = g_2^{(t)} + g_2^{(t+1)} + a_m. \]

\[ \square \]

Next, we will show that if \( t < n/4 \) then the algorithm will find \( g_p^{(t)} \) and \( g_h^{(t)} \) with no (non-trivial) short-paths between them.

**Lemma 99.** If \( t < n/4 \), then on the \( t \)-th iteration the algorithm will find \( g_p^{(t)} \) and \( g_h^{(t)} \) with no (non-trivial) short-paths between them.

**Proof.** Fix some \( g_p^{(t)} \). By Claim 39, for each \( G_x(G) \), there exists \( g_h^{(t)} \) such that there is a short-path between \( g_p^{(t)} \) and \( g_h^{(t)} \). Therefore \( g_p^{(t)} \) has \( n \) different short-paths. Since \( t < n/4 \) or \( 2t < n/2 \), there are at least \( n/2 + 1 \) options for choosing \( g_h^{(t)} \), and each of them has a trivial short-path with \( g_p^{(t)} \). Therefore, we are left with \( n/2 - 1 \) short-paths, and at least \( n/2 + 1 \) column vectors \( g_h^{(t)} \). Thus, there is at least one of them that has no (non-trivial) short-path with \( g_p^{(t)} \). \[ \square \]

203
We are now ready to conclude with the following theorem.

**Theorem 100.** For $\tau = n/4$ the algorithm $\text{FBSolution2}(\tau, M)$ will construct recovery sums for the first $2^{s-2}$ requests of $M$.

**Proof.** Since $\tau = n/4$, by Lemma 99, on the $t$-th iteration the algorithm $\text{FB-Solution2}(\tau, M)$ will find $g_p^{(t)}$ and $g_h^{(t)}$ with no (non-trivial) short-paths between them. Therefore, by Claim 40, after executing Steps 4 and 5, the columns $g_{2t}^{(t)}$ and $g_{2t+1}^{(t)}$ have no (non-trivial) short-paths between them. By Lemma 98, since there are no (non-trivial) short-paths between $g_{2t}^{(t)}$ and $g_{2t+1}^{(t)}$, by the end of this iteration it holds that

$$g_{2t}^{(t+1)} + g_{2t+1}^{(t+1)} = v_t,$$

and all the pair sums $g_{2t'}^{(t+1)} + g_{2t'+1}^{(t+1)}$ such that $t' < t$ will be unchanged. \qed

**Remark 2.** By using this technique we can obtain a solution for the case in which $M$ has exactly two kinds of vectors ($d = 2$). Suppose that $v_1$ appears $t_1$ times and $v_2$ appears $t_2$ times in $M$, and assume that $t_1 \geq t_2$. Then, initialize $G$ such that each pair of columns will be summed up to $v_1$ and use this technique to update the first $t_2 \leq n/4$ pairs to $v_2$.

We proceed to the second case, i.e., $\tau \leq \lfloor n/3 \rfloor$.

**6.6.2 The $\tau \leq \lfloor n/3 \rfloor$ Case**

If $n/4 < t \leq \lfloor n/3 \rfloor$, then the algorithm may not be able to find $g_p^{(t)}$ and $g_h^{(t)}$ with no (non-trivial) short-paths between them. However, at least one of these pairs will have at most one (non-trivial) short-path between them. Therefore, we may not be able to make some requests on the $t$-th iteration and reach Step 17. However, if we are left to handle more than one kind of request in $M$, we will succeed in this iteration. For that, we present the $\text{BadCaseCorrection}(G^{(t)}, a_t, M)$ function.

---

**BadCaseCorrection($G^{(t)}, a_t, M$)**

1: Let $C_i \in C_{a_t}(G)$ be a cycle s.t. $\{g_{2t}^{(t)}, g_{2t+1}^{(t)}\}_p \in C_i$
2: Find $\{g_{2t'}^{(t)}, g_{2t'+1}^{(t)}\}_p \in C_i$ s.t. $g_{2t'}^{(t)} + g_{2t'+1}^{(t)} \neq v_t$
3: $j \leftarrow \text{FindShortPath}(G^{(t)}, a_t, t, t')$
4: $G^{(t)} \leftarrow \mathcal{F}_{a_t}(g_{2t+1}^{(t)}, g_{2t}^{(t)})$
5: Swap $g_{2t'}^{(t)}$ and $g_{2t}^{(t)}$
6: Swap $g_{2t'+1}^{(t)}$ and $g_{2t+1}^{(t)}$
7: Swap $v_t$ and $v_{t'}$ in $M$
8: Return $G^{(t)}, M$

---

204
Lemma 101. On the $t$-th iteration such that $t < \lfloor n/3 \rfloor$, there exist $p,h \geq 2t$ such that there is at most one (non-trivial) short-path between $g^{(t)}_p$ and $g^{(t)}_h$, and by the end of this iteration it holds that

$$g^{(t+1)}_{2t} + g^{(t+1)}_{2t+1} = v_m,$$

and $t$ recovery sums are satisfied.

Proof. Fix some $g^{(t)}_p$. We already proved that $g^{(t)}_p$ has $n$ different short-paths. Since $t < \lfloor n/3 \rfloor$ or $2t < \lfloor 2n/3 \rfloor$, there are at least $\lceil n/3 \rceil + 1$ options for choosing $g^{(t)}_h$, and each one of them has a trivial short-path with $g^{(t)}_p$. Therefore, we left with $\lfloor 2n/3 \rfloor - 1$ short-paths and $\lceil n/3 \rceil + 1$ column vectors $g^{(t)}_h$. Therefore, in the worst case, there is a column vector $g^{(t)}_h$ such that there is exactly one (non-trivial) short-path $P_x$ between $g^{(t)}_h$ and $g^{(t)}_p$. If $x \neq g^{(t)}_p + g^{(t)}_h + v_t$, Step 12 will succeed, and we can construct a recovery set of size 2 for $v_t$ on the $t$-th iteration of the $FBSolution2(\tau,M)$ algorithm, as proved in Lemma 98. If $x = g^{(t)}_p + g^{(t)}_h + v_t$, we cannot obtain $v_t$ on the $t$-th iteration. So from now we assume that $x = g^{(t)}_p + g^{(t)}_h + v_t$.

Assume that we have at least two different requests $v_i,j \geq t$ in $M$, and let $v_m \neq v_t$, for $m \geq t$. We prove that there does not exist a short-path $P_y$ such that $y = g^{(t)}_p + g^{(t)}_h + v_m$. Therefore, Step 12 will succeed to find $\{g^{(t)}_{2\ell}, g^{(t)}_{2\ell+1}\}_p \in C_i$ such that $\ell > t$. As proved in Lemma 98 the algorithm $FBSolution2(\tau,M)$ will construct a recovery set of size 2 for $v_m$ on the $t$-th iteration.

We are left with considering the case where all the requests $v_m$ in $M$ for $m \geq t$ are identical (and thus $v_m = v_t$). Also assume that $g^{(t)}_{2t} + g^{(t)}_{2t+1} \neq v_t$ which means that Step 3 has failed (otherwise this iteration will succeed). In this case the algorithm will use its $BadCaseCorrection(G^{(t)}, a_m, M)$ function. Let $C_i \in C_{a_m}(G)$ be a cycle such that $\{g^{(t)}_{2t}, g^{(t)}_{2t+1}\}_p \in C_i$ (Step 1). First assume that for every pair-type edge $\{g^{(t)}_{2t}, g^{(t)}_{2t+1}\}_p \in C_i$ such that $t' < t$ it holds that

$$g^{(t)}_{2t'} + g^{(t)}_{2t'+1} = v_{t'}.$$

Note that if a path

$$g^{(t)}_{2t'} - g^{(t)}_{2t'+1} - g^{(t)}_{2t} - g^{(t)}_{2t+1}$$

holds

$$g^{(t)}_{2t'} + g^{(t)}_{2t'+1} = g^{(t)}_{2t} + g^{(t)}_{2t+1} = v_t,$$

then it has to be a cycle of length 4. Therefore, in our case,

$$g^{(t)}_{2t} + g^{(t)}_{2t+1} = g^{(t)}_{2t'} + g^{(t)}_{2t'+1} = v_t.$$
which is a contradiction to the assumption that \( g_{2t}^{(t)} + g_{2t+1}^{(t)} \neq v_t \). Otherwise, we can assume that there is an edge \( \{g_{2t'}, g_{2t'+1}\}_p \in C_i, t' < t \) such that

\[
g_{2t'}^{(t)} + g_{2t'+1}^{(t)} = v_{t'} \neq v_t.
\]

By executing Step 4, \( g_{2t'}^{(t)} + g_{2t'+1}^{(t)} \) will be updated to \( v_{t'} \), which corrupts the sum of the pair \( \{g_{2t'}, g_{2t'+1}\}_p \) such that

\[
g_{2t'}^{(t)} + g_{2t'+1}^{(t)} = v_{t'} + a_t \neq v_{t'}.
\]

After executing Steps 5–6, we obtain two kinds of requests in \( M \) that are left to deal with, while still having \( t - 1 \) valid recovery sets. In this case, the algorithm will return to Step 3. Since now we have two kinds of requests, as already proved, the algorithm will be able to construct a recovery set for either \( v_t \) or \( v_{t'} \) on the \( t \)-th iteration.

The next theorem follows directly from Lemma 101.

**Theorem 102.** If \( \tau \leq \lfloor n/3 \rfloor \), then the algorithm \( FBSolution2(\tau, M) \) will construct recovery sums for the first \( \lfloor \frac{2}{3} \cdot 2^s - 1 \rfloor \) requests of \( M \).

Due to Theorem 102, we proved that the algorithm \( FBSolution2(\tau, M) \) provides an alternative construction for \( FB-(2^s - 1, s, \lfloor \frac{2}{3} \cdot 2^s - 1 \rfloor) \) codes. However, the algorithm \( FBSolution2(\tau, M) \) is better than the algorithm \( FBSolution(\tau, M) \) since by using the algorithm \( FBSolution2(\tau, M) \), all the recovery sets are of length 2 (and not 4). Therefore, we are left with \( \lceil n/3 \rceil \) unused column vectors. If (6.7) holds, we will put \( \lfloor n/3 \rfloor - 2 \) of these columns, except the redundant columns \( g_{n-2}, g_{n-3} \), as the first columns of the \( HG \)-matrix \( G \), and similarly, the left requests of \( M \) that have no recovery sets yet are placed first. In the next section, we show how to obtain \( \lfloor \frac{1}{6} \cdot 2^s - 1 \rfloor + 1 \) more recovery sets of size (at most) 4 from these \( \lceil n/3 \rceil \) unused columns of \( G \).

### 6.6.3 Constructing Recovery Sets of Size 4

According to the previous results as stated in Theorem 74(c), and due to (6.7), we can assume that \( s \geq 7 \). The \( FBSolution(\tau, M) \) algorithm uses the initialized \( HG \)-matrix \( G^{(0)} \) that satisfies (6.2). In fact, we can construct a similar algorithm that is initialized by any arbitrary \( HG \)-matrix \( G \). Let \( \tau = \lfloor \frac{1}{6} \cdot 2^s - 1 \rfloor - s \). The value of \( \tau \) represents the number of requests that will be handled. Note that

\[
4\tau = 4\left\lfloor \frac{1}{6} \cdot 2^s - 1 \right\rfloor - 4s
= 4\left\lfloor \frac{1}{12} \cdot 2^s \right\rfloor - 4s \leq \left\lfloor \frac{1}{3} \cdot 2^s \right\rfloor - 2.
\]
for $s \geq 7$, which is the number of unused columns in $G$. Our goal is to use either 2 or 4 columns of $G$ for every recovery set. In other words, every $v_t$ will be equal to either $g_{4t} + g_{4t+1}$ or $g_{4t} + g_{4t+1} + g_{4t+2} + g_{4t+3}$. To show this property we will prove that in every step of the algorithm, we have to have at least $2(s+1)$ unused (or redundant) columns of $G$.

We start with the next definition, which is based on the fact that every $s + 1$ vectors in $F_2^s$ have a subset of $h \leq s + 1$ linearly dependent vectors.

**Definition 103.** Given an $HG$-matrix $G^{(t)}$, denote the set $S_h^{(t)} \subseteq [s + 1]$ of size $h \leq s + 1$, such that

$$\sum_{i \in S_h^{(t)}} \left( g_{4t+2i}^{(t)} + g_{4t+2i+1}^{(t)} \right) = 0.$$  

Denote the $Reorder(G^{(t)})$ procedure that swaps arbitrarily between the columns of $G^{(t)}$ presented in (6.8) and the columns indexed by $\{4t \leq m \leq 4t + 2h - 1\}$ in $G^{(t)}$ and returns the reordered matrix and $h$ as an output.

By using the $Reorder(G^{(t)})$ procedure which is defined in Definition 103, we can assume that

$$\sum_{i \in [h]} \left( g_{4t+2i}^{(t)} + g_{4t+2i+1}^{(t)} \right) = 0.$$  

We are now ready to show the $FBSolution3(G, \tau, M)$ algorithm, which is presented as Algorithm 6.

**Algorithm 6 $FBSolution3(G, \tau, M)$**

1: $G^{(0)} \leftarrow G$
2: $B^{(0)} = \emptyset$
3: for $t = 0, 1, \ldots, \tau - 1$ do
    4: $G^{(t)}, h \leftarrow Reorder(G^{(t)})$
    5: $G^{(t)} \leftarrow FindEquivSums(G^{(t)}, t, h)$
    6: $G^{(t+1)}, B^{(t+1)} \leftarrow FindGoodOrBadRequest(G^{(t)}, 2t, B^{(t)}, v_t)$
4: end for
8: Return $G^{(\tau)}$ and $B^{(\tau)}$

Note that since

$$\left\lfloor \frac{1}{3} \cdot 2^s \right\rfloor - 2 - 4\tau = \left\lfloor \frac{1}{3} \cdot 2^s \right\rfloor - 4 \left\lfloor \frac{1}{12} \cdot 2^s \right\rfloor + 4s - 2 \geq 4s - 2 \geq 2(s + 1),$$  

for $s \geq 7$, the $2h \leq 2(s + 1)$ column vectors of $G^{(t)}$ presented in (6.9) are unused on the $t$-th iteration. By using these $2h$ unused columns, the function $FindEquivSums(G, t, h)$ will be able to reorder the columns of $G^{(t)}$ such that

$$g_{4t}^{(t)} + g_{4t+1}^{(t)} = g_{4t+2}^{(t)} + g_{4t+3}^{(t)},$$  

(6.10)
without changing the previous valid recovery sums. Then, the function \( \text{FindGoodOrBadRequest}(G^{(t)}, 2t, B^{(t)}, v_t) \) will update the sum \( g_{4t}^{(t)} + g_{4t+1}^{(t)} \) to either \( v_t \) or \( g_{4t}^{(t)} + g_{4t+1}^{(t)} + v_t \), again, without changing all the previous valid recovery sums. In the latter case, we are able to construct a recovery set \( R_t = \{4t, 4t + 1, 4t + 2, 4t + 3\} \) of size 4 due to (6.10).

The \( \text{FindEquivSums}(G, t, h) \) algorithm is presented next.

\[
\text{FindEquivSums}(G, t, h).
\]

1: for \( i = 1, \ldots, h - 1 \) do
2: \( x_i \leftarrow g_{4t+2i} + g_{4t+2i+1} \)
3: if \( g_{4t} + g_{4t+1} \neq x_i \) then
4: \( G \leftarrow \text{FindGoodOrBadRequest}(G, 2t, B x_i) \)
5: end if
6: if \( g_{4t} + g_{4t+1} = x_i \) then
7: Swap \( g_{4t+2} \) and \( g_{4t+2i} \)
8: Swap \( g_{4t+3} \) and \( g_{4t+2i+1} \)
9: Return \( G \)
10: end if
11: end for

The proof of the correctness of the function \( \text{FindEquivSums}(G, t, h) \) is shown in the following theorem.

**Theorem 104.** If there are at least \( 2(s + 1) \) unused columns in \( G \), then there is a function \( \text{FindEquivSums}(G, t, h) \) that can reorder the columns of \( G \) such that

\[
g_{4t} + g_{4t+1} = g_{4t+2} + g_{4t+3},
\]

without corrupting the previous valid recovery sums.

**Proof.** As explained before, the \( 2h \leq 2(s + 1) \) column vectors of \( G \) presented in (6.3) are unused. Therefore, the algorithm \( \text{FindEquivSums}(G, t, h) \) will not corrupt the previous valid recovery sums. Our goal is to prove that the algorithm will succeed on Step 6. On the \( i \)-th iteration, by executing Step 4, the sum of \( g_{4t} + g_{4t+1} \) will be either \( g_{4t} + g_{4t+1} + x_i \) or \( x_i \), without changing other sums except of the redundancy sum \( g_{n-2} + g_{n-1} \). If \( g_{4t} + g_{4t+1} = x_i \), Step 6 will succeed. Otherwise, we can assume that on the \( i \)-th iteration, the algorithm obtains \( g_{4t} + g_{4t+1} + x_i \) on Step 4. Denote by \( x_0 \) the sum of \( g_{4t} + g_{4t+1} \) at the beginning of the algorithm. Therefore, at the end of the \( i \)-th iteration, we obtain

\[
g_{4t} + g_{4t+1} = \sum_{j=0}^{i} x_j.
\]

By (6.9),

\[
\sum_{j=0}^{h-2} x_j = x_{h-1}.
\]
Therefore, on the last iteration, i.e., when \( i = h - 1 \),
\[
g_{4t} + g_{4t+1} = \sum_{j=0}^{h-2} x_j = x_{h-1}.
\]

Thus, Step 3 will fail and Step 6 will succeed, concluding the proof.

We are ready to show the main result of this section.

**Lemma 105.** The \( FBSolution3(G, \tau, M) \) algorithm constructs \( \tau \) valid recovery sets \( R_t \) for \( v_t \), without corrupting previous recovery sums.

**Proof.** By Theorem 104, the \( \text{FindEquivSums}(G^{(t)}, t, M) \) algorithm will output \( G^{(t)} \) such that \( g_{4t}^{(t)} + g_{4t+1}^{(t)} = g_{4t+2}^{(t)} + g_{4t+3}^{(t)} \), and all previous sums of requests are valid. Before the execution of the \( \text{FindGoodOrBadRequest}(G^{(t)}, 2t, B^{(t)}, v_t) \) function we denote the sums \( g_{4t} + g_{4t+1} \) and \( g_{4t+2} + g_{4t+3} \) by \( y_t \). The algorithm \( \text{FindGoodOrBadRequest}(G^{(t)}, 2t, B^{(t)}, v_t) \) will update only the sum \( g_{4t} + g_{4t+1} \) to either \( v_t \) or \( v_t + y_t \) and the sum of the last pair \( g_{n-2} + g_{n-3} \) which is redundant, due to Lemma 85. The case \( g_{4t} + g_{4t+1} = v_t \) is called a good case and we assume that all such \( t \)'s are inserted in a set \( G \). These pairs will be recovered by the recovery sets \( R_t = \{4t, 4t+1\} \). The case \( g_{4t} + g_{4t+1} = v_t + y_t \) is called a bad case and all such \( t \)'s are assumed to be inserted in a set \( B \). By (6.10) for every \( t \) such that \( g_{4t} + g_{4t+1} = v_t + y_t \) we have that \( g_{4t+2} + g_{4t+3} = y_t \). Thus, in these cases, the requests will have the recovery sets \( R_t = \{4t, 4t+1, 4t+2, 4t+3\} \).

In Section 6.5, we showed a technique to obtain another recovery set from the redundancy pair \( g_{n-2} \) and \( g_{n-1} \). Using this technique, we are able to construct \( \left\lfloor \frac{1}{6} \cdot 2^{s-1} \right\rfloor - s + 1 \) valid recovery sets. By combining the three cases above, the following theorem is deduced immediately.

**Theorem 106.** An \( FBs-(2^s - 1, s, \left\lfloor \frac{5}{6} \cdot 2^{s-1} \right\rfloor - s) \) code exists.

### 6.7 A Construction of \( Bs-(2^s - 1, s, 2^{s-1}) \) Codes

Wang et al. [12] showed a construction for \( Bs-(2^s - 1, s, 2^{s-1}) \) codes, which is optimal, using a recursive decoding algorithm. In this section, we show how to achieve this result with the simpler, non-recursive decoding algorithm. Our solution solves even a more general case in which the requests \( v_j \)'s satisfy some constraint that will be described later in this section. The idea of this algorithm is similar to the one of the \( FBSolution(\tau, M) \) algorithm. First, we slightly change the definition of a Hadamard solution as presented in Definition 77 to be the following one.
Definition 107. Let \( M = [v_0, v_1, \ldots, v_{n/2-1}] \) be a request of order \( s \times n/2 \), where \( n = 2^k \). The matrix \( M \) has a Hadamard solution if there exists an HG-matrix \( G = [g_0, g_1, \ldots, g_{n-1}] \) of order \( s \times n \) such that for all \( i \in [n/2 - 1] \),

\[
v_i = g_{2i} + g_{2i+1}, \]

and for \( i = n/2 - 1 \) either \( v_i = g_{n-2} + g_{n-1} \), or \( v_i = g_{n-2} \), or \( v_i = g_{n-1} \). In this case, we say that \( G \) is a Hadamard solution for \( M \).

Let \( G \) be an HG-matrix. Let \( \mathbf{G} \) be the set of all matrices \( G' \) generated by elementary row operations on \( G \). The following claim proves that elementary row operations on HG-matrices only reorder their column vectors.

Claim 41. Every \( G \in \mathbf{G} \) is an HG-matrix.

Proof. We will only prove that adding a row in \( G \) to any other row, generates an HG-matrix. By proving that, it can be inductively proved that doing several such operations will again yield an HG-matrix.

Without loss of generality, we assume that we add the \( i \)-th row, for some \( 0 < i \leq s - 1 \), to the 0-th row of \( G \) and generate a new matrix \( G' \). Assume to the contrary that \( G' \) is not an HG-matrix. Thus, there are two distinct indices \( \ell, m \in [n] \) such that \( g'_\ell = g'_m \). Therefore, by definition of elementary row operations, \( G \) satisfies \( g_\ell = g_m \), which is a contradiction. \( \square \)

Let \( M \) be a request denoted by

\[
M = [v_0, v_1, \ldots, v_{2^s-1-1}].
\]

Let \( \mathbf{M} \) be the set of all matrices \( M' \) generated by elementary row operations on request matrix \( M \). We now present Lemma 108. Its proof follows directly from Claim 41.

Lemma 108. If there is an \( M \in \mathbf{M} \) such that there is a Hadamard solution for \( M \), then there is a Hadamard solution for all \( M' \in \mathbf{M} \).

Proof. Let \( M \in \mathbf{M} \) and let \( G \) be a Hadamard solution for \( M \). Let \( P \) be a set of elementary row operations, generating \( M' \) from \( M \). By Claim 41, executing elementary row operations \( P \) on \( G \) generates an HG-matrix, \( G' \). Since we applied the same elementary row operations \( P \) on both \( M \) and \( G \), it is deduced that \( G' \) is a Hadamard solution for \( M' \). \( \square \)

The constraint mentioned above is as follows. Given \( \mathbf{M} \), we demand that there is a request \( M' \in \mathbf{M} \) having the 0-th row to be a vector of ones. Using Lemma 108, our algorithm will handle any request \( M' \) such that \( M' \in \mathbf{M} \) and \( \mathbf{M} \) holds this constraint. Note that if each request vector \( v_j \in \mathbb{F}_2^s \) is a unit vector, then by summing up all of its rows to the 0-th one, it holds that there exists such
a matrix in $M$ holds the constraint. Moreover, if every request vector is of odd Hamming weight, our algorithm will still find a solution. Therefore, from now on, we assume that the 0-th row of the request matrix $M$ is a vector of ones.

Remember that $e = (0, 0, \ldots, 0, 1) \in \mathbb{F}_2^n$. The initial state of the matrix

$$G = [g_0, g_1, \ldots, g_n]$$

will satisfy

$$g_{2t} + g_{2t+1} = e, \quad t \in [n/2].$$

Given $g \in \mathbb{F}_2^n$, let $f(g)$ be the bit value in the 0-th position in $g$. Remember that for all $x \in \mathbb{F}_2^n$, the graph $G_x(G)$ has a partition to $\ell \geq 1$ disjoint simple cycles, that will be denoted by $C_\ell(G) = \{C_i\}_{i=0}^{\ell-1}$ (Definition 81). Let $\tau = 2^{s-1}$. We are now ready to show the following algorithm.

\begin{algorithm}
\begin{itemize}
\item [1:] $G^{(0)} \leftarrow G$
\item [2:] for $t = 0, \ldots, \tau - 2$ do
\item [3:] if $f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 1$ then
\item [4:] Find $g_p^{(t)}, g_h^{(t)}$ s.t. $p, h \geq 2t$ and $f(g_p^{(t)} + g_h^{(t)}) = 0$
\item [5:] Swap $g_p^{(t)}$ and $g_{2t}^{(t)}$
\item [6:] Swap $g_h^{(t)}$ and $g_{2t+1}^{(t)}$
\item [7:] end if
\item [8:] $a_t \leftarrow v_t + g_{2t}^{(t)} + g_{2t+1}^{(t)}$
\item [9:] Let $C_i \in C_{a_t}(G^{(t)})$ be a cycle s.t. $\{g_{2t}^{(t)}, g_{2t+1}^{(t)}\} \in C_i$
\item [10:] Find $\{g_{2m}^{(t)}, g_{2m+1}^{(t)}\} \in C_i$ s.t. $m > t$
\item [11:] $j \leftarrow \text{FindShortPath}(G^{(t)}, a_t, t, m)$
\item [12:] $G^{(t+1)} \leftarrow \mathcal{F}_{a_t}(g_{2t}^{(t)}, g_{j}^{(t)})$
\item [13:] end for
\item [14:] if $g_n^{(\tau-1)} + g_{n-1}^{(\tau-1)} \neq v_{n/2-1}$ and $g_{n-2}^{(\tau-1)} \neq v_{n/2-1}$ then
\item [15:] $G^{(\tau)} \leftarrow \mathcal{F}_{g_n^{(\tau-1)} + v_{n/2-1}}(g_{n-2}^{(\tau-1)}, g_{n-1}^{(\tau-1)})$
\item [16:] end if
\item [17:] Return $G^{(\tau)}$
\end{itemize}
\end{algorithm}

Our first goal is to prove that on the $t$-th iteration when the $BSolution(\tau, M)$ algorithm reaches Step 8, it holds that $f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 0$.

**Lemma 109.** On the $t$-th iteration when the $BSolution(\tau, M)$ algorithm reaches Step 8, it holds that $f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 0$.

**Proof.** If on the $t$-th iteration in Step 8 $f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 1$ then the $BSolution(\tau, M)$ algorithm will try to find $p, h \geq 2t$ such that $f(g_p^{(t)} + g_h^{(t)}) = 0$. Note

211
that since \( t \leq \tau - 2 \), we have \( g_{2t}^{(t)}, g_{2t+1}^{(t)} \) and at least two more column vectors \( g_{2m}^{(t)}, g_{2m+1}^{(t)} \) such that \( m > t \). By the pigeonhole principle, there exist two indices \( p, h \in \{2t, 2t+1, 2m, 2m+1\} \) such that \( f(g_{p}^{(t)}) = f(g_{h}^{(t)}) \). After executing Step 5 and Step 6 we obtain \( f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 0 \).

Our next goal is to show that in Step 10 on the \( t \)-th iteration, the \textsc{BSolution}(\( \tau, M \)) algorithm will find \( \{g_{2m}^{(t)}, g_{2m+1}^{(t)}\}_{p} \in C_{i} \) such that \( m > t \). We start with the following claim.

**Claim 42.** Given a graph \( G_{x}(G) \) and its partition to cycles \( C_{x}(G) = \{C_{i}\}_{i=0}^{p-1} \), for all \( C_{i} \in C_{x}(G) \) it holds that

\[
\sum_{\{g_{2m}, g_{2m+1}\}_{p} \in C_{i}} (g_{2m} + g_{2m+1}) \equiv \frac{1}{2} |C_{i}| x (\mod 2).
\]

**Proof.** Assume that \( C_{i} \) is of length \( 2\ell \), and its cycle representation is given as follows

\[ C_{i} = g_{s_{0}} - g_{s_{1}} - \cdots - g_{s_{2\ell-1}} - g_{s_{2\ell}} - g_{s_{0}}, \]

where both of the edges \( \{g_{s_{0}}, g_{s_{1}}\}_{p}, \{g_{s_{2\ell-1}}, g_{s_{2\ell}}\}_{p} \) are pair type edges. By Claim 34, for all odd \( t \in [2\ell] \) it holds that \( g_{s_{t}} = g_{s_{t+1}} + x \). Thus, by summing only the sums of the nodes of the pair-type edges in \( C_{i} \) we obtain

\[
\sum_{\{g_{2m}, g_{2m+1}\}_{p} \in C_{i}} (g_{2m} + g_{2m+1})
= \sum_{t \in [2\ell]} g_{s_{t}} = \sum_{t \in [2\ell], t \text{ is odd}} (g_{s_{t}} + g_{s_{t}} + x)
\equiv \ell x (\mod 2).
\]

Now we are ready to prove the following lemma.

**Lemma 110.** On the \( t \)-th iteration when the \textsc{BSolution}(\( \tau, M \)) algorithm reaches Step 10, it will find \( \{g_{2m}^{(t)}, g_{2m+1}^{(t)}\}_{p} \in C_{i} \) such that \( m > t \).

**Proof.** Remember that we assumed that the bit value in the 0-th position for all the requests \( v_{j} \) is 1. Therefore, on the \( t \)-th iteration, for all \( m < t \) it holds that \( f(g_{2m}^{(t)} + g_{2m+1}^{(t)}) = 1 \), and by Lemma 109 \( f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 0 \). Now, assume to the contrary that there are no \( m > t \) such that \( \{g_{2m}^{(t)}, g_{2m+1}^{(t)}\}_{p} \in C_{i} \). Note that

\[
f(a_{t}) = f(v_{t} + g_{2t}^{(t)} + g_{2t+1}^{(t)})
= f(v_{t}) + f(g_{2t}^{(t)} + g_{2t+1}^{(t)})
= 1 + 0 \equiv 1 (\mod 2).
\]

212
By Claim \[\text{Claim 42}\] if \(|C_i| = 2\ell\) then,

\[
\sum_{\{g_{2m}^{(t)}, g_{2m+1}^{(t)}\} \in C_i} (g_{2m}^{(t)} + g_{2m+1}^{(t)}) \equiv \ell a_i (\text{mod } 2),
\]

and therefore,

\[
\sum_{\{g_{2m}^{(t)}, g_{2m+1}^{(t)}\} \in C_i} f(g_{2m}^{(t)} + g_{2m+1}^{(t)}) \equiv \ell f(a_i) = \ell (\text{mod } 2).
\]

However, since only the edge \(\{g_{2t}^{(t)}, g_{2t+1}^{(t)}\} \in C_i\) satisfies that \(f(g_{2t}^{(t)} + g_{2t+1}^{(t)}) = 0\), it is deduced that

\[
\sum_{\{g_{2m}^{(t)}, g_{2m+1}^{(t)}\} \in C_i} f(g_{2m}^{(t)} + g_{2m+1}^{(t)}) = \ell - 1 \not\equiv \ell (\text{mod } 2),
\]

which violates Claim \[\text{Claim 42}\].

We are ready to show the main theorem of this section.

**Theorem 111.** Given a request matrix \(M\) having the \(0\)-th row to be a vector of ones, the \(\text{BSolution}(\tau, M)\) algorithm finds a Hadamard solution for \(M\).

**Proof.** First, we will prove that the \(\text{BSolution}(\tau, M)\) algorithm generates \(2^{s-1} - 1\) recovery sets for the first \(2^{s-1} - 1\) requests \(v_t\). This is done by Steps \[\text{Steps 1–12}\]. Note that the sums \(g_{2m}^{(t)} + g_{2m+1}^{(t)}\) for all \(m < t\), might be changed only after Step \[\text{Step 10}\].

We will show that these sums will not be changed and the sum \(g_{2t}^{(t)} + g_{2t+1}^{(t)}\) will be equal to \(v_t\) at the end of the \(t\)-th iteration. By Lemma \[\text{Lemma 110}\], when the \(\text{BSolution}(\tau, M)\) algorithm reaches Step \[\text{Step 10}\], it will find \(\{g_{2m}^{(t)}, g_{2m+1}^{(t)}\} \in C_i\) such that \(m > t\). Thus, there is a good-path between \(g_{2t+1}\) and one of the nodes \(g_{2m}, g_{2m+1}\) (the closest one between them to \(g_{2t+1}\)), and the index of this node is denoted by \(j\) (Step \[\text{Step 11}\]). Due to Lemma \[\text{Lemma 110}\], by executing \(F_{a_i}(g_{2t+1}, g_j)\), the matrix \(G^{(t)}\) is updated to a matrix \(G^{(t+1)}\) such that only the two following pair summations are correctly changed to

\[
\begin{align*}
g_{2t}^{(t+1)} + g_{2t+1}^{(t+1)} &= g_{2t}^{(t)} + g_{2t+1}^{(t)} + a_t = v_t \\
g_{2m}^{(t+1)} + g_{2m+1}^{(t+1)} &= g_{2m}^{(t)} + g_{2m+1}^{(t)} + a_t.
\end{align*}
\]

Lastly, Steps \[\text{Steps 14–15}\] handle the last recovery set in a similar way as was done in the proof of Theorem \[\text{Claim 42}\].

### 6.8 Conclusion

In this paper, functional \(k\)-batch codes and the value \(\text{FB}(s, k)\) were studied. It was shown that for all \(s \geq 6\), \(\text{FB}(s, \lfloor \frac{s}{6} 2^{s-1} \rfloor - s) \leq 2^s - 1\). In fact, we believe that
by using a similar technique, this result can be improved to \( \lfloor \frac{7}{5} 2^{s-1} \rfloor - s \) requests, but this proof has many cases and thus it is left for future work. We also showed a family of \( FB(2^s + \lfloor (3\alpha - 2) \cdot 2^{s-2} \rfloor - 1, s, \lfloor \alpha \cdot 2^{s-1} \rfloor) \) codes for all \( 2/3 \leq \alpha \leq 1 \). Yet another result in the paper provides an optimal solution for \( k = 2^s \) which is \( FB(s, 2^{s}) = 2^s + 1 - 2 \). While the first and main result of the paper significantly improves upon the best-known construction in the literature, there is still a gap to the conjecture which claims that \( FB(s, 2^{s-1}) = 2^s - 1 \). We believe that the conjecture indeed holds true and it can be achieved using Hadamard codes.

Appendix A

Claim 33. For any triple-set \((\mathcal{G}, \mathcal{B}, \mathcal{R})\) if \(|\mathcal{B}| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor \) then \(|\mathcal{B}| \leq |\mathcal{R}|\).

Proof. According to the definition of \((\mathcal{G}, \mathcal{B}, \mathcal{R})\) and since \( \mathcal{G} \cup \mathcal{B} = \lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor \) it holds that

\[
|\mathcal{R}| = |2^{s-1} \setminus (\mathcal{G} \cup \mathcal{B} \cup \{2^{s-1} - 1\})| = 2^{s-1} - \lfloor \frac{2}{3} \cdot 2^{s-1} \rfloor - 1.
\]

Thus, in order to prove that \(|\mathcal{B}| \leq |\mathcal{R}|\), since \(|\mathcal{B}| \leq \lfloor \frac{1}{3} \cdot 2^{s-1} \rfloor\), we will prove inequality \((a)\) in

\[
|\mathcal{R}| = 2^{s-1} - \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor - 1 \geq \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor \geq |\mathcal{B}|.
\]

This inequality is equivalent to

\[
2^{s-1} - 1 \geq \left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor + \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor.
\]

We separate the proof for the following two cases.

Case 1: If \( s \) is even, then \( 2^s \equiv 1 \pmod{3} \), \( 2^{s-1} \equiv 2 \pmod{3} \).

Thus,

\[
\left\lfloor \frac{2}{3} \cdot 2^{s-1} \right\rfloor + \left\lfloor \frac{1}{3} \cdot 2^{s-1} \right\rfloor = \left\lfloor \frac{2^s}{3} \right\rfloor + \left\lfloor \frac{2^{s-1}}{3} \right\rfloor = \frac{2^s - 1}{3} + \frac{2^{s-1} - 2}{3} = \frac{3 \cdot 2^{s-1} - 3}{3} = 2^{s-1} - 1.
\]

Case 2: If \( s \) is odd, then \( 2^s \equiv 2 \pmod{3} \), \( 2^{s-1} \equiv 1 \pmod{3} \).
Thus,

\[
\left\lfloor \frac{2}{3} \cdot 2^s - 1 \right\rfloor + \left\lfloor \frac{1}{3} \cdot 2^s - 1 \right\rfloor = \frac{2^s}{3} + \left\lfloor \frac{2^{s-1}}{3} \right\rfloor
= \frac{2^s - 2}{3} + \frac{2^{s-1} - 1}{3} = \frac{3 \cdot 2^{s-1} - 3}{3} = 2^{s-1} - 1.
\]

Therefore, it is deduced that in both cases if \(|B| \leq \left\lfloor \frac{1}{3} \cdot 2^s - 1 \right\rfloor\) then \(|B| \leq |R|\).
Bibliography


Discussion
Chapter 7

Discussion

This dissertation discussed codes over graphs. It is based on my papers which were published and submitted to the main journals and conferences in the fields of coding theory and information theory. This topic combines the theory and practicality of computer sciences.

First, we studied a new class of codes, called codes over graphs. This model assumes that there are undirected complete graphs with $n$ nodes (vertices) and the information is stored on the undirected edges which connect every two nodes in the graph. Under this setup, there are $\binom{n}{2}$ edges in the graph which store $\binom{n}{2}$ symbols over some alphabet $\Sigma$. Such codes are constructed to deal with node failures, i.e., an erasure of whole neighborhoods of arbitrary $\rho$ nodes. We then extended this model to directed graphs and construct codes over directed complete graphs. Under this setup, there are $n^2$ edges in the graph which store $n^2$ symbols over some alphabet $\Sigma$.

Another new family of codes is called codes over trees. Each codeword in such a code will be a tree with a fixed number of nodes. A unique form of a tree, i.e., a topology and an arrangement of its nodes, is an information that we want to store and read. Such codes will be constructed with the ability to reconstruct trees with erroneous or erased edges. We discussed some constructions and bounds on the size of such codes. We also investigated the asymptotic size of tree balls of trees of radius $t$.

The last subject discussed in this thesis deals with constructions for functional $k$-batch codes. This study is relevant to the so called parallel RIO codes, that are relatively connected to non-volatile (flash) memories.

7.1 Codes over Graphs

A code over directed graphs is called a directed $\rho$-node-erasure-correcting code if it can correct the failure of any $\rho$ nodes in each graph in the directed code. An undirected $\rho$-node-erasure-correcting code is defined similarly. The minimum redundancy $r_D$, $r_U$ of any directed, undirected $\rho$-node-erasure-correcting code of
length \( n \), satisfies

\[
\begin{align*}
r_D & \geq n^2 - (n - \rho)^2 = 2n\rho - \rho^2, \\
r_U & \geq \left( \frac{n + 1}{2} \right) - \left( \frac{n - \rho + 1}{2} \right) = n\rho - \left( \frac{\rho}{2} \right),
\end{align*}
\]

respectively. A code over directed, undirected graphs satisfying the first, second inequality with equality will be called \textit{optimal}, respectively. Note that optimal codes can be represented systematically by placing the information on the edges of the first \( k \) nodes. Hence, for systematic codes over graphs, the number of redundancy nodes is at least \( \rho \). Note that for all \( n \) and \( \rho \), one can always construct an optimal directed \( \rho \)-node-erasure-correcting code from an \([n^2, (n - \rho)^2, 2n\rho - \rho^2 + 1]\) MDS code. Similarly, one can always construct an optimal undirected \( \rho \)-node-erasure-correcting code from an \([\binom{n+1}{2}, \binom{n-\rho+1}{2}, n\rho - \binom{\rho}{2} + 1]\) MDS code. However, in both cases, the field size of the graph codes will be at least \( \Theta(n^2) \).

In paper \cite{1} we presented several constructions that reduce the field size to \( \Theta(n) \). Namely, we show a construction of optimal \textit{\( \rho \)-node-erasure-correcting code} over \( \mathbb{F}_q \) where \( q \) is a prime power such that \( q \geq |\Sigma| \). This result is developed by constructing product codes that were introduced by Elias in \cite{1} and generalized by Tanner in \cite{8}.

In the binary case, we have general sub-optimal constructions for both directed and undirected cases. For the directed case, we encode \( n^2 - 2n\rho \) information bits into a graph with \( n \) nodes while tolerating \( \rho < n/2 \) node failures. Note that this construction does not provide optimal \( \rho \)-node-erasure-correcting codes since \( r_D = 2n\rho \), which does not meet the bound in \eqref{eq:7.1}. For example, for \( \rho = 2 \) the difference between the code redundancy and the bound is 4 redundancy bits. For the undirected case we encode

\[
k = \begin{cases} \frac{n(n - 2\rho + 1)}{2}, & \text{if } n \text{ is odd,} \\ \frac{(n + 1)(n - 2\rho)}{2}, & \text{if } n \text{ is even,} \end{cases}
\]

information bits into a graph with \( n \) nodes while tolerating \( \rho < n/2 \) node failures. Again, this construction does not provide optimal \( \rho \)-node-erasure-correcting codes since

\[
r_U = \begin{cases} n\rho, & \text{if } n \text{ is odd,} \\ (n + 1)\rho, & \text{if } n \text{ is even,} \end{cases}
\]

which does not meet the bound in \eqref{eq:7.2}. For example, for \( \rho = 2 \) the difference between the code redundancy and the bound is one redundancy bit for odd \( n \) and three redundancy bits for even \( n \).

Our next goal in this research was to achieve the two main properties simultaneously, that is, optimal and binary codes over graphs. We showed a construction for double node failures for both directed and undirected cases, when the number of nodes is a prime number. We also provided two efficient encoding and decoding algorithms for both constructions. By defining a minimal graph distance for
these codes, we proved that they are of minimum graph distance 3. Therefore, these codes are capable of correcting a single error of a node.

We also showed some important bounds on codes over graphs. The first says that encoding 3 information bits into $n$ nodes, while tolerating $\rho = n - 2$ node failures over $\mathbb{F}_q$ should satisfy $q^2 + q + 2 > n$. In that case, the number of such codes is

$$q^2 \binom{n}{2}(q-1)^{\frac{n+1}{2}}(q^2 + q + 1)! \left(\frac{q^2 + q + 1}{q^2 + q + 1 - n}\right)!.$$

We also showed more general upper bound in case that there are $\binom{k+1}{2}$ information bits (that can be represented by $k$ information nodes), while tolerating $\rho = n - k$ node failures over $\mathbb{F}_q$.

In paper [12] we presented another binary optimal construction for double-node-erasure correction together with an efficient decoding algorithm, when the number of nodes is a prime number. Furthermore, we extended this construction for triple-node-erasure-correcting codes when the number of nodes is a prime number and two is a primitive element in $\mathbb{Z}_n$. These codes are at most a single bit away from optimality. Finding an optimal construction for this case is left for future work. It is also interesting to find a construction that tolerates three node failures for the directed case.

Another result followed the recent works on regenerating codes [7, 24, 29] in order to analyze a sufficient number of edges we have to read in order to correct a single node erasure while using our double-node-erasure-correcting code. We showed that in order to correct a single node erasure, we are not required to read the rest of the graph in its entirety. Namely, while the number of edges in the graph is $\frac{n(n+1)}{2}$, we showed that it is enough to read only $\frac{5}{12} n^2 + O(n)$ edges in order to decode a single node failure.

### 7.2 Codes over Trees

We initiated the study of codes over trees and the tree distance. Upper bounds on such codes were presented together with specific code constructions for several parameters of the number of nodes and minimum tree distance.

Every tree over $n$ nodes can be represented by a binary vector of length $\binom{n}{2}$ called the characteristic vector. Such a vector is indexed by all possible $\binom{n}{2}$ edges that the tree can have and it has ones only in the indices of the tree’s edges. Using this representation, the tree distance between any two trees is one half the Hamming distance between their characteristic vectors. Thus, the tree distance is a metric. A code with tree distance $d$ can correct $d - 1$ edge erasures. The largest size of a code over trees with tree distance $d$ is denoted by $A(n, d)$. Codes over trees achieving $A(n, d)$ are called optimal.

We showed three upper bounds. The first bound is a sphere packing bound. The second, third bound is an improved upper bound in case that $d = n - 2, d = n - 3$.
1. For all $n \geq 1$ and fixed $d$, 
\[
A(n, d) \leq F(n, d) / \binom{n-1}{d-1} = O(n^{n-1-d}).
\]

2. For all positive integers $n$, $A(n, n - 2) \leq n$.

3. For all $n \geq 9$, $A(n, n - 3) \leq n^2$.

We then showed another approach to obtain both lower and upper bounds on codes over trees using tree balls of trees of radius $t$ denoted by $V_T(n, t)$. For that, tree balls of trees of radius one are studied and the main results on these balls are

1. For any $T \in T(n)$, 
\[
V^*(n, 1) \leq V_T(n, 1) \leq V^-(n, 1).
\]

2. For all $n \geq 1$, $V(n, 1) \approx 0.5 \sqrt{\frac{2}{n}} n^{2.5} = \Theta(n^{2.5})$.

3. For all $n \geq 1$, $V^*(n, 1) = \Theta(n^2)$, $V^-(n, 1) = \Theta(n^3)$.

We could calculate these ball sizes due to the special form of stars and path trees. We then showed that $V_T(n, 1) \leq V^-(n, 1)$, and then improved it for arbitrary $t$. By using the fact that $V^*(n, 1) = \Theta(n^2)$, the upper bound $A(n, 3) = O(n^{n-4})$ is concluded which is the same upper bound result as the sphere packing bound. Applying the generalized Gilbert-Varshamov bound, while using the fact that $V(n, 1) = \Theta(n^{2.5})$, it is then deduced that $A(n, 2) = \Omega(n^{n-4.5})$.

We then showed that for all $T \in T(n)$ and fixed $t$, it holds that

1. $V_T(n, t) = \Omega(n^{2t})$, $V_T(n, t) = O(n^{3t})$.

2. $V(n, t) = \Theta(n^{2.5t})$.

3. $V^*(n, t) = \Theta(n^{2t})$ and $V^-(n, t) = \Theta(n^{3t})$.

These results are derived from recursive formulas that calculate the size of the tree balls of trees of radius $t$.

Again, using the fact that $V^*(n, t) = \Theta(n^{2t})$, it is deduced that for all $d = 2t + 1$, $A(n, d) = O(n^{n-1-d})$ which matches the upper bound results by the sphere packing bound. Applying the generalized Gilbert-Varshamov lower bound and using the fact that $V(n, t) = \Theta(n^{2.5t})$, it is also derived that for $d = t + 1$, $A(n, d) = \Omega(n^{n-2-2.5(d-1)})$.

Next we showed the sizes of tree balls of trees of stars and path trees for arbitrary radius. Our main contribution in this section is formulated in recursive formulas for the sizes of tree balls of trees for arbitrary trees. We then showed
upper and lower bounds on these formulas using the sizes of tree balls of trees of the star and path trees. For all \( n \) and fixed \( t \) let

\[
P = n^{t-1} \binom{n-1}{t} (n-t), \quad Q = n^{t-1} \binom{n+t}{2t+1}.
\]

The following properties holds:

1. \[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V^*(n,t-i) = P.
\]

2. \[
\sum_{i=0}^{t} \binom{n-2-t+i}{i} V^-(n,t-i) = Q.
\]

3. For all \( T \in \mathbf{T}(n) \)

\[
P \leq \sum_{i=0}^{t} \binom{n-2-t+i}{i} V_T(n,t-i) \leq Q.
\]

We then conjectured that for fixed \( t \) and \( n \) large enough,

\[
V^*(n,t) \leq V_T(n,t) \leq V^-(n,t).
\]

Lastly, we provided several constructions that improve upon the generalized Gilbert-Varshamov lower bounds. The results of these constructions are

1. There exists an \( \mathcal{T}_-(n, \lfloor n/2 \rfloor, n-1) \) code.

2. There exists an \( \mathcal{T}_-(n, n, n-2) \) code.

3. For any positive integer \( d \leq n/2 \), there exists an \( \mathcal{T}_-(n, M, d) \) code such that \( M = \Omega(n^{n-2d}) \).

4. For fixed \( m \) and prime \( n \), there exists an \( \mathcal{T}_-(n, \frac{n-1}{2} \cdot \lfloor \frac{n-1}{m} \rfloor, \lfloor \frac{3n}{4} \rfloor - \lfloor \frac{3n}{2m} \rfloor - 2) \) code.

The result 4. assures that it is possible to construct codes of cardinality \( \Omega(n^2) \), while the minimum distance \( d \) approaches \( \lfloor 3n/4 \rfloor \) and \( n \) is a prime number. Comparing to \( A(n, n-3) = \mathcal{O}(n^2) \), this result shows that \( A(n, d) = \Omega(n^2) \), when \( d \) approaches \( \lfloor 3n/4 \rfloor \) and \( n \) is prime. Thus, finding the range of values of \( d \) for which \( A(n, d) = \Theta(n^2) \) is left for future work.

While the results in the paper provide a significant contribution in the area of codes over trees, there are still several interesting problems which are left open. Some of them are summarized as follows.
1. Improve the lower and upper bounds on the size of codes over trees, that is, the value of $A(n, d)$.

2. Find an optimal construction for $d = n - 3$.

3. Study codes over trees under different metrics such as the tree edit distance.

4. Study the problem of reconstructing trees based upon several forests in the forest ball of trees; for more details see [3].

7.3 Functional $k$-Batch Codes

The functional $k$-batch codes and the value $FB(s, k)$ were studied in this thesis. In particular, a construction of $FB((2^s - 1, s, \lfloor \frac{2}{3} \cdot 2^s - 1 \rfloor)$ codes is given. Moreover, we showed how to construct $FB((2^s + [(3 \alpha - 2) \cdot 2^{s-2} - 1, s, \lfloor \alpha \cdot 2^s - 1 \rfloor)$ codes for all $2/3 \leq \alpha \leq 1$. Then, we improved it to $FB(s, \lfloor \frac{5}{8} 2^s - 1 \rfloor - s)$ for $s \geq 6$. In fact, we believe that by using a similar technique, this result can be improved to $\lfloor \frac{7}{8} 2^s - 1 \rfloor - s$ requests, but this proof has many cases and thus it is left for future work.

Another result that can be found in [34] states that $FP(s, 2^s) \leq 2^{s+1} - 2$. In this case, the lower bound is the same, i.e., this result is optimal, see [10]. We showed that this optimality holds not only for functional PIR codes but also for the more challenging case of functional batch codes, that is, $FB(s, 2^s) = 2^{s+1} - 2$. We achieved all these results using a generator matrix $G$ of a Hadamard codes of length $2^s$ and dimension $s$, where the matrix’s columns correspond to the servers of the $FB-(n, s, k)$ code.

Another direction is to find construction where there are a fixed $d$ number of requests $v_j$. In this thesis we remarked that for $d = 2$, it can be solved by one of our constructions, and for $d = 3$ we believe we have a proof which is not a part of this work.

We also mentioned that Wang et al. [31] showed a construction for $B-(2^s - 1, s, 2^s - 1)$ codes, which is optimal, using a recursive decoding algorithm. We showed how to achieve this result with the simpler, non-recursive decoding algorithm. This solution is stronger since it solves a more general case in which the requests $v_j$’s are of odd Hamming weight.

While the first and main result significantly improves upon the best known construction in the literature, there is still a gap to the conjecture which claims that $FB(s, 2^s - 1) = 2^s - 1$. We believe that the conjecture indeed holds true and it can be achieved using Hadamard codes.
Bibliography


[24] N. Raviv, N. Silberstein, and T. Etzion, “Constructions of high-rate mini-
mum storage regenerating codes over small fields,” Proceedings of the IEEE


[26] E. Sharon and I. Alrod, “Coding scheme for optimizing random I/O perfor-


[28] K.-U. Schmidt, “Symmetric bilinear forms over finite fields of even charac-
teristic,” J. of Combinatorial Theory, Series A, vol. 117, no. 8, pp. 1011–1026,
May 2010.


[30] M. Vajha, V. Ramkumar, and P. V. Kumar, “Binary, shortened project-
ive Reed Muller codes for coded private information retrieval,” In Proceed-


associative memories,” Proceedings of the IEEE International Symposium on
2012.

using coset coding with Hamming code,” In Proceedings IEEE Information

[34] Y. Zhang, T. Etzion, and E. Yaakobi, “Bounds on the length of functional
PIR and batch codes,” IEEE Transactions on Information Theory, vol. 66,

[35] G. Zhou and M. Gen, “Approach to the degree-constrained minimum span-
ning tree problem using genetic algorithm,” Engineer design and automation,
We satisfy the following condition for each $k$ in the requests of linear combinations $v_0, v_1, \ldots, v_{k-1}$ (not necessarily distinct) of information bits, there exist $k$ groups of reconstruction $R_0, R_1, \ldots, R_{k-1} \subseteq \{0,1, \ldots, n-1\}$ satisfying $\sum_{j \in R_i} y_j = v_i$ for each group $R_i$. We denote $FB(s, k)$ the minimum number of information bits for $FB - (n, s, k)$.

For $s \leq 5$ characteristic functions $FB = (2^s-1, s, 2^{s-1})$ we use the characteristic function $n = 2^s - 1$ of $s$ characteristic bits. For $s \geq 6$, we denote $k = 2^s - 2 + 2^{s+4} \leq \frac{2^s}{\sqrt{2^4}}$ the characteristic function. The characteristic function for the characteristic function $FB = (n, s, k)$ is $FB = \left(\frac{2}{3} \cdot 2^{s-1}\right)$. We denote $k = [a2^{s-1}]$, where $a \leq 1$ is a constant and $n \geq 2^s - 1$.

For $s \geq 6$, we denote $k = \left[\frac{5}{6}2^{s-1}\right] - s$, where $n = 2^s - 1$.

For $s \geq 6$, we denote $k = 2^s - s$, where $n = 2^s - 1$.

For $s \geq 6$, we denote $k = 2^{s+1} - 2$, where $n = 2^s - 1$.

For $s \geq 6$, we denote $k = \left[\frac{5}{6}2^{s-1}\right] - s$, where $n = 2^s - 1$.

For $s \geq 6$, we denote $k = \left[\frac{5}{6}2^{s-1}\right] - s$, where $n = 2^s - 1$.
\[ \Omega(n^{n-2d}) \leq A(n,d) \leq O(n^{n-d-1}) \]

\[ \Theta(n^2) \leq V_T(n, t) \leq \Theta(n^{3/2}) \]

\[ M = \Omega(n^{n-2d}) \cdot d \leq n/2 \]

\[ \text{binary shift} \text{ to } \lfloor \frac{n-1}{m} \rfloor \cdot \left( \lfloor \frac{3n}{2m} \rfloor - 2 \right) \text{ }

\text{otherwise.} \]

\[ T = (n, n, n - 2) \]

\[ T = (n, n, n - 2) \]

\[ \text{binary shift to } \lfloor \frac{n-1}{m} \rfloor \cdot \left( \lfloor \frac{3n}{2m} \rfloor - 2 \right) \text{ }

\text{otherwise.} \]

\[ T = (n, n, n - 2) \]

\[ T = (n, n, n - 2) \]

\[ \text{binary shift to } \lfloor \frac{n-1}{m} \rfloor \cdot \left( \lfloor \frac{3n}{2m} \rfloor - 2 \right) \text{ }

\text{otherwise.} \]

\[ T = (n, n, n - 2) \]
In order to represent the data structure in a computer, we usually define the nodes as memory cells. For data structures, we use trees to store information in a certain hierarchy. When we want to unite one, we avoid errors. Therefore, we define another question: how many of the edges we can add. There are edge representations of trees, which can be used in families of trees (even if up to a fixed number of nodes). One can represent trees as trees, and we can use edge representations of trees for data structure storage. For example, the trees of parsing trees, syntax trees (AST) or trie are used to represent nodes in the graph. We define the number of nodes by the maximum number of messages at each node. Therefore, the number of nodes in the graph is defined as follows. First, we use the dictionary and the vector representation of a field, which is used to define the number of nodes in the graph. Then we use the dictionary representation of the graph to define the number of messages at each node. Finally, we use the dictionary representation of the graph to define the number of messages at each node. Therefore, we can define the number of nodes in the graph as follows.

Let's define the number of nodes in the graph as a function of the number of edges. For example, the number of nodes in the graph is defined as $\frac{n}{2}$. Therefore, we can define the number of nodes in the graph as follows.
تقدיר

כידוע, בתורת הצפנה הקלאסית מידע המיוצג על ידי וקטורים מעל אלפבית קבוע. מהלך זה שימושי ברוב מערכות במדעי המחשב, אך לא תמיד מתאים לכל מערכות המידע הקיימות. למשל, מדענים פיתחו צופני תיקון שגיאות במטריצות. צופני אלו נועדו لتיקון מחיקות של שורות ועמודות במטריצות הממימדים קבועים ומעל אלפבית קבוע. בנוסף, ישנן עבודות רבות על צופנים שבהן מילים הקוד הן תמורות באורך קבוע. צופנים אלו נועדו لتיקון תמורות פגומות, כלומר, תמורות שเซ almacen ומשדר תמים ביןvaluate באלפבית קבוע. בעבודה זו אנו מציגים משפחה חדשה של צופנים שנקראת צופן לשחזור צמתים בגרפים. מודל זה נועד לשחרור פעמים נוספים מידע על מידע של פונקציונליות של המוח המאחסן ומפענח מידע על ידי אסוציאציות בין תאים הזיכרון בו. יתר על כן, בעזרת ייצוג מידע על ידי גרפים ניתן ליצור מודל של מערכות מבואות המאחסן מידע שנשמר בין כל זוג צמתים במערכת. לדוגמה, הצמתים יכולים לשימור משתמשים, והקשתות יכולות לשימור אינפורמציה שנשמרה בין כל זוג משתמשים. במקביל, נרחבות המרחב של אינפורמציה.getAddress משמשות במערך זה.

צופנים לשחזור גרפים מוגדרים באופן הבא. מילת צופן מוגדרת בתור גרף מלא עם לולאות עצמות ועם מספר קבוע של צמתים \( n \). האינפורמציה בצלופנים אלו מוגדרת על ידי סימובולים מעלי אלפבית קבוע \( \Sigma \), ואני מורה על הקשתות של графיקות בלבד. אנו מגדירים שתי משפחות שונות של צופנים לש�回 צמתים בגרפים, צופן לש�回 צמתים מכוונים וצופן לש姮 צמתים בלתי מכוונים. צופן לש姮 צמתים מכוונים הוא צופן שמכיל אך ורק גרפים מכוונים מלאים עם לולאות עצמות. בדומה לצופן לש姮 צמתים מכוונים, צופן לש缑 צמתים בלתי מכוונים הוא צופן שמכיל אך ורק גרפים בלתי מכוונים מלאים עם לולאות עצמות.

מחיקה של צומת מוגדרת בתור מחיקה של קבוצה של קשתות המחוברות לצומת מסוים, בהנחה שמספר הצומת שנמחק ידוע למפענח. נאמר כי הצופן לש 하나님의 צמתים ברグラפים הוא בעל יכולת תיקון של \( \rho \) מחיקות, אם הצלופן יכול לתקן כל מחיקה של \( \rho \) צמתים כלשהם בגרף.

עבור המקרה המכוון, מספר הקשתות בגרף \( n \). עבור המקרה בלתי מכוון, מספר הקשתות \( 2n - \rho \).

במידה שהגרפים בלתי מכוונים, מספר הקשתות \( r \) שדרוש למילוי \( \rho \) מחיקות הוא:\n
\[ r \geq \frac{n}{2} - \left( \frac{n - \rho + 1}{2} \right) = \nu \rho - \frac{\rho^2}{2} \]

בעבודה זו אנו מציגים משפחה של צופנים אופטימליים המתאימים לצופן התיקון \( \rho \) מחיקות, \( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשר למצוא את המספר \( \rho \) בשיטה שמצインターハ\( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשרのではない \( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשרのではない \( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשרのではない \( \nu \) צומתים, \( \sigma \) אינפורמציה. אפשרのではない \( \nu \) צומתים, \( \sigma \) אינפורмещенיה. אפשרنجح \( \nu \) צומתים, \( \sigma \) אינפורмещенיה. אפשרنجح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינפורمنهجיה. אפשרنجاح \( \nu \) צומתים, \( \sigma \) אינفور
תודה ל榷בי ממעקהולתה, יוני לווש, רחל ברטו, איתי פירברקר, יונתן בוכניק.шеוקלח, יוני שופר, יובל אפרת, ומרט בר, שוהים אחינו וליהום זה ועמדנו לא רך.
במחבר, אלא גם בחתי היה-זימיר.

נדע לה któשים המוחות, שעמגמה עלייתו לאורים כל התgons. התgons ורעה. התgons את נהנה
ולבכלган. הנח על האבים ול耢ד הבוקפות הפעמים. ולעגון לרען זה. אח דידי
מהמשת לארוך כל土耳其. תודה לפשפחת שלuario השיעותלה בניו.

תודה立ちית המקוונים, אחנה חנה על האבה והמענה защиты, ילידות víctima פרתראץיו הוהים
יכשורי חיים, הגרלה ואחי להחיי עליילולשת דיברים בטיבור, ילידות והארה
דיברים על צום שዞברה לא יתיייווה.

תודה והGetValueות לכל אנשי היקרים ילשלאדרס ולא יתיי מנייל. תודה לכב!
הpolator הביעęk ב old name של פרופ' איתן יעקובי, בפקולת מחשבים בטכניון, וה rdrkk.

תודה

ברצוני להודות לבראstructor של הפרופ' איתן יעקובי, בראשות הקוללת למדעי המחשב בטכניון. הפרופ' איתן יעקובי הסבירו לי את הערוץ ואת המאבטח של העברת חותם למחקר ול_way של מקצועיות ו어서 מצו.

אני אסיר תודה למניחי המאמר איתן יעקובי אשר הדרי את המחקר של מעידות במקצועיות ובאנושיות. הפרופ' איתן יעקובי הוא המנהל האأجر המרכזית מיום תחילת מחקרי

אני אסיר תודה לבראstructor עם שילוי עולם ששלח לי את האנשים הנכונים, הרעיונות, הכח והיכולת לכתוב תזה זו. אני אסיר תודה למניחי המאמר איתן יעקובי אשר הדרי את המחקר של מעידות במקצועיות ובאנושיות. הפרופ' איתן יעקובי הוא המנהל האジャー המרכזית מיום תחילת מחקרי

אני COMPANY על תמיכות, ייעודים וקוודים מפסיע ראשית בעולם האקדמי ועד לעצמי המחקרית. דאגתי لي למלגות ושא الزمن, אף יותר מאשר דאגתי לעצמי. תודה שתנתתי לי ללמוד מכם בכל תחומים, ואני מבטיח להשתדל ליישם.

המון תודה לקבוצת המחקרים והחברה של חוג מחקרי בפקולטה, והנה החברים המקסימים שפגשתי בפקולטה, אלה החברים Wilhelm ש總是 התעניינה, תמכו ברגעים הקשים, והفقد את הורים לי את הדרך ב妁ים יפים וنعימים.

בפרט אודה לבראstructor בקורס "קומבינטוריקה למדעי המחשב" שבנויית בירושלים ובבראstructor הראשית פרופ' איתן יעקובי, פרופ' טובי עציון, פרופ' רוני רוט, פרופ' נאדר בשותי ופרופסór שמואל זקס. אני חב לכם documentos על תמיכה והכוונה לפיתוח אישי.

אני COMPANY על תמיכות, ייעודים וקוודים מפסיע ראשית בעולם האקדמי ועד לעצמי המחקרית. דאגתי לי למלגות ושא الزمن, אף יותר מאשר דאגתי לעצמי. תודה שתנתתי לי ללמוד מכם בכל תחומים, ואני מבטיח להשתדל ליישם.

אני COMPANY על תמיכות, ייעודים וקוודים מפסיע ראשית בעולם האקדמי ועד לעצמי המחקרית. דאגתי לי למלגות ושא الزمن, אף יותר מאשר דאגתי לעצמי. תודה שתנתתי לי ללמוד מכם בכל תחומים, ואני מבטיח להשתדל ליישם.

אני COMPANY על תמיכות, ייעודים וקוודים מפסיע ראשית בעולם האקדמי ועד לעצמי המחקרית. דאגתי לי LMLimenti ושא الزمن, אף יותר מאשר Dאגתי לעצמי. תודה שתנתתי לי ללמוד מכם בכל תחומים, ואניебיא להשתדל ליישם.
צרפונים לשחזרת גרפים

תורמים על מחקר

לישם מלאי חלקי של תרדירות לקבילה ולהואלה
דוקטור לפילוסופיה

לב יוחננוב

הוגשפורס المنطقة – מכון טכנולוגי לישראל
אדר החשף בחיפה פברואר 2022
 الزوجים לעולם גוף

לב יוהננס