Graphs and Self-dual additive codes over $GF(4)$

Mithilesh Kumar, Srimathi Varadharajan, and Håvard Raddum

Simula UiB, Bergen, Norway
mithilesh@simula.no, srimathi.varadharajan@uib.no, haavardr@simula.no

Abstract. We initiate the study of self-dual codes over $GF(4)$ whose corresponding graphs have fixed rankwidth. We show that by combining the structural properties of rankwidth 1 graphs, the classification of corresponding codes becomes exponentially faster. We present a new algorithm for computing weight enumerators using Binary Decision Diagrams (BDD), which is an essential step in the classification of the codes. In addition, we show that the minimum distance of a code is at least 3 if and only if the corresponding graph does not contain any pendant vertex or any twin-pairs. We also present an algorithm for computing an approximate minimum distance of codes corresponding to general graphs.
Keywords— Stabilizer Code, Self-dual, Rankwidth, Binary Decision Diagram, Minimum Distance

1 Introduction

In cryptography, self-dual additive codes over $GF(4)$ (of type IV) have been used in the construction of secret sharing schemes as they provide higher security level over linear codes [29, 4, 19]. Self-dual codes have strong connection to quantum error correction [14, 5] and measurement-based quantum computation [25, 18]. These codes have been used in the construction of unimodular lattices [16, 17]. As lattices are interesting in cryptography, this provides additional motivation to study self-dual codes. There has been a lot of work done in the classification of these codes since early 70s [23]. Combinatorially, there is a one-to-one correspondence between self-dual additive codes over $GF(4)$ and simple undirected graphs [31, 21, 3, 28]. There is a long list of papers [12, 1, 8, 9, 6, 7] that have tried to classify such codes using this relation. But most of these approaches have tried to focus scantly on the richness graph theory can bring to the study of such codes.

Trying to classify codes by considering all possible graphs makes the problem extremely hard. This is evident since self-dual additive codes over $GF(4)$ has been classified only for n up to 12 [8]. It is natural to restrict the study to specific graph classes in the hope that the classification of the corresponding codes can be pushed further for larger values of n. In this paper, we initiate the study of self-dual codes whose corresponding graphs have fixed rankwidth, where rankwidth is a graph property giving non-negative integer values. For example, the class of graphs with rankwidth 1 are exactly the distance-hereditary graphs. This choice is motivated by the fact that two self-dual codes over $GF(4)$ are equivalent

if and only if the corresponding graphs are related by local complementation and the graph parameter rankwidth is preserved under local complementation. We show that by combining the structural properties of these graphs with the algorithm used in [13, 8], the classification of the corresponding codes becomes exponentially faster. In the above branching algorithm, for graphs of rankwidth 1, instead of branching in $2^n - 1$ ways, we need only branch in $3n - 3$ ways.

There are two computationally heavy steps in the above algorithm: graph isomorphism and weight enumeration. For a fixed positive integer k, testing graph isomorphism for graphs of rankwidth k is polynomial in the size of the graph [15], and it is in fact linear in n for graphs of rankwidth 1 [30]. Hence, looking at the problem of classification of such codes in terms of rankwidth has additional advantages.

Another important step in the classification algorithm from [8] was computing weight-enumerators for a given code. The algorithm given in [8] to compute the weight-enumerator for a linear $[n, k]$ code is essentially a brute-force search with complexity $\mathcal{O}(2^k)$. If $k > n/2$, it is necessary to go via the dual code to calculate the weight enumerators. We provide a new unified approach using Binary Decision Diagrams (BDD) to compute the weight-enumerators.

The minimum distance of codes corresponding to distance hereditary graphs is 2. We show that the minimum distance of a code is at least 3 if and only if the corresponding graph does not contain any pendant vertex or any twin-pairs. We also present an algorithm for computing an approximate minimum distance of codes corresponding to general graphs and leave some interesting open problems.

2 Preliminaries

We use standard graph theory notation. A *graph* is a pair $G = (V, E)$ where V is a set of *vertices*, and $E \subseteq V \times V$ is a set of *edges*. A graph with n vertices can be represented by an $n \times n$ binary matrix called the *adjacency matrix* A such that $A_{ij} = 1$ if and only if $v_i v_j \in E$, and $A_{ij} = 0$ otherwise. The *open neighbourhood* of $v \in V$, denoted by $N(v)$, is the set of vertices connected to v by an edge. The *closed neighbourhood* of $v \in V$, denoted by $N[v]$, is the set of vertices connected to v by an edge along with v itself. The number of vertices incident to v is called the degree of a vertex v. The *complement* or the *inverse* of a graph G is a graph H such that two distinct vertices of H are adjacent if and only if they are not adjacent in G. A vertex of degree 1 is called a *pendant*. A pair of vertices u, v are called *true twins* if $N[u] = N[v]$ and are called false twins if $N(u) = N(v)$. When we call a pair of vertices as *twin-pair*, then they can either be true-twins or false-twins. *Local complementation* (LC) on v denoted by $G * v$ replaces the induced subgraph of G on $N(v)$ by its complement. See Figure 1. A *graph code* is an additive code over GF(4) that has a generator matrix of the form $C = A + \omega I$, where I is the identity matrix and A is the adjacency matrix of a simple undirected graph. A graph code is self-dual since its generator matrix has full rank over GF(2).

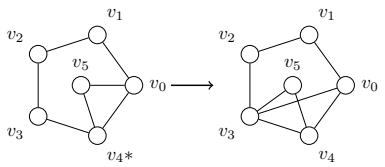

Fig. 1: Local complementation at v_4

Theorem 1 ([31],[21]). *Every self-dual additive code over GF(4) is equivalent to a graph code.*

2.1 Rankwidth

Let G be a simple undirected graph. A rank decomposition of G is a pair (T, L) of tree T with leaves L such that every internal node in the tree has degree 3 and there is a bijection between the set of leaves L of T to the set of vertices V of G. See Figure 2. For every edge e in T, we can associate a partition (A, B) of

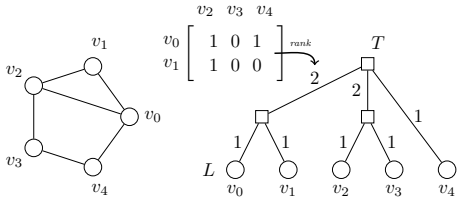

Fig. 2: A rank decomposition (T, L) of a graph. The rankwidth of T is 2. It so happens that the rankwidth of this graph is 2 as well.

V, where A and B are set of leaves in the two sub-trees that result after deleting e. The weight of an edge e in T is the rank of the adjacency matrix of bipartite graph $((A, B), E)$ where E is the set of edges whose one endpoint is in A and other in B. The rankwidth of the decomposition (T, L) is the largest weight of an edge in T. The rankwidth $rw(G)$ of G is the smallest rankwidth of any tree decomposition of G. Local complementation preserves the rankwidth of a graph.

Theorem 2 ([22]). *Given a graph G and $v \in V(G)$, $rw(G) = rw(G * v)$.*

The *distance* between two nodes is the length of the shortest path between them. A distance-hereditary graph is a graph in which the distances in any connected induced subgraph are the same as they are in the original graph.

Theorem 3 ([22]). *G is distance-hereditary if and only if the rankwidth of G is at most* 1.

2.2 Binary Decision Diagrams

The data structure we use for calculating the weight enumerators for self dual additive code over $GF(4)$ are Binary Decision Diagrams (BDD). Binary Decision Diagrams are used in various applications, such as representing system of Boolean equations [27], application to permutations [20], or representing integer multiplication [24]. In this paper, we describe the basic operations on a BDD in order to compute the weight enumerators of self dual codes of $GF(4)$.

Description of BDD A Binary Decision Diagram (BDD) is a directed acyclic graph with a root node at the top and a true-node at the bottom. The nodes in a BDD are arranged in horizontal *levels*, and we visualize a BDD by drawing the levels in a top-down fashion.

All edges in the BDD are directed downwards, with an edge always going between nodes on different levels. In other words, no edge is drawn between the nodes on the same level. Each node, except for the bottom node, has one or two outgoing edges, called the *0-edge* and/or the *1-edge*. The bottom node has only incoming edges. 0-edges are drawn as dotted lines, while 1-edges are drawn as solid lines. All the operations on BDD are done over GF(2). The transition from GF(4) to GF(2) is explained in Section 3.

A level in a BDD is usually associated with a single variable over GF(2). In our case, we allow a *linear combination* of variables over GF(2) associated with each level. A *path* in a BDD is a sequence of consecutive edges, where the end node of one edge is the start node for the next edge. A *complete* path starts in the top node and ends in the bottom node. We regard each edge in a path to assign a value over GF(2). If an e-edge starts from a node on a level associated with linear combination l, it yields the linear equation $l = e$.

Adding and Swapping Levels The add operation allows us to add one linear combination for a level onto the linear combination for the level directly below, and to change the BDD accordingly to keep the set of binary vectors encoded by the BDD unchanged. The add operation was first explained in [27], and the basic operation done for each node on a level is shown in Figure 3.

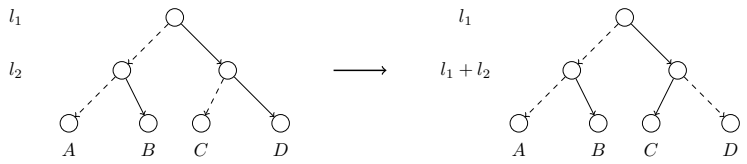

Fig. 3: Adding levels in a BDD

How to swap the variables on two adjacent levels in a BDD and change the nodes and edges such that the resulting BDD encodes exactly the same set of

vectors was first explained in [26]. By swapping levels the linear combination associated with level i is swapped to level $i+1$ and vice versa without affecting the set of vectors encoded by the BDD. The basic general operation of swapping levels is shown in Figure 4.

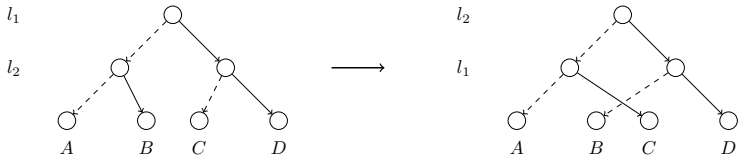

Fig. 4: Swapping levels in a BDD

3 Construction of BDD

Let G be an $n \times n$ generator matrix of a code C over $GF(4)$. We compute the set of all code words by considering all possible linear combinations over $GF(2)$ of the rows of G. This is done by first expanding the matrix into an $n \times 2n$ matrix G' over $GF(2)$ by associating (mapping) each $GF(4)$-element to two bits as follows: $0 = (00)$, $1 = (01)$, $\omega = (10)$, $\omega^2 = (11)$.

$$G = \begin{bmatrix} \omega & 1 \\ 1 & \omega \end{bmatrix}_{n \times n} \Longrightarrow G' = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}_{n \times 2n}$$

Now we multiply all binary strings $(c_1, c_2, c_3, ...c_n)$ of length n to the matrix G' to get the set of all code words $(x_1, x_2, x_3, ...x_{2n-1}, x_{2n})$.

In order to construct the BDD that has all code words as paths, we introduce the parity check matrix H. The parity check matrix describes the set of linear relations that the coordinates of each code word must satisfy. If x is a code word and H is the parity check matrix then $xH^T = 0$.

Let the linear equations given by $xH^T = 0$ be $l_i = 0$ for $1 \leq i \leq n$. It is now easy to construct a BDD that encodes all code words of the code, i.e. x-vectors satisfying all $l_i = 0$. Start by listing $l_1, \ldots, l_n$ as the linear combinations for the top n levels. Each level has a single node with a 0-edge going to the node on the level below. This ensures that only x-vectors satisfying $l_i = 0, i = 1, \ldots, n$ are encoded in the BDD. The bottom n levels have n "free" variables as associated linear combinations, in the sense that all free variables are linearly independent from each other and the l_i's. The node on the level of l_n has a 0-edge that jumps over the bottom n levels, going straight to the bottom node.

We now use add and swap operations on this basic BDD to resolve the linear combinations l_i. By resolving the linear combinations we mean that we add together some of the linear combinations and free variables to transform l_i into a single variable. We use the swap operation to move levels that need to be added so they are adjacent to each other, before doing the add operation.

We apply the operations until only single variables appear on all levels. We sort the levels in order such that x_1 appears on the top and x_{2n} appears on the lowest level.Now the paths of the BDD represent all code words in C.

Complexity: In this paper we are concerned with the special case where the codes are of length $2n$ and dimension n. However, constructing the BDD representing a binary code can be done for any length n and any dimension $k \leq n$. We give the complexity, in terms of number of nodes in the final BDD, in the general case for an $[n, k]$ linear code C over $GF(2)$.

Lemma 1. *The number of nodes on any level of the final BDD after resolving all linear combinations for a code C is at most 2^k.*

Lemma 2. *The number of nodes on any level of the final BDD after resolving all linear combinations for a code C is at most 2^{n-k}.*

Combining lemmas 1 and 2 we get the following result.

Theorem 4. *The number of nodes in the final BDD representing the code words of a binary linear $[n, k]$ code is of order $\mathcal{O}(2^{min\{k,n-k\}})$.*

3.1 Algorithm for computing weight enumerator

Recall that pairs of coordinates (x_{2i-1}, x_{2i}) actually represent one element in $GF(4)$. A path in the BDD with resolved and sorted levels has length $2n$, but represents a code word of length n with elements from $GF(4)$. When computing the weight enumeration we therefore count how many non-zero $GF(4)$-elements a path (code word) represents.

We proceed to describe our algorithm for computing weight enumerators using BDDs. Assume that we have constructed the BDD representing a $GF(4)$-code, with the levels having $x_1, \ldots, x_{2n}$ as linear combinations.

1. Start with setting $(1, 0, 0, \ldots, 0)$ as the vector for the true-node at the bottom. We say there is one path of weight 0 from the true-node to itself.
2. We compute the vectors for the other nodes in a recursive way, from the lower levels to higher ones. When a pair of edges from a node T to A contribute 0 to the path weight, the weight distribution below T along this path is the same as for A. In other words, prepending the partial code words with a zero does not change the weight distribution. When the pair of edges from T to A contribute 1 to the weight, the paths of weight i below A become paths of weight $i + 1$ below T. Hence the weight enumeration vectors for T are obtained by shifting the vector for A by one position to the right.
3. Assuming all weight distribution vectors have been computed for the nodes on one level, compute the weight distribution for the nodes two levels above by adding all the weight contributions, shifting them by one position to the right as needed. This is shown in Figure 5.
4. Compute weight distributions for all nodes in the BDD, moving upwards two levels at the time. In the end, the vector for the root node gives the weight distribution of the whole code.

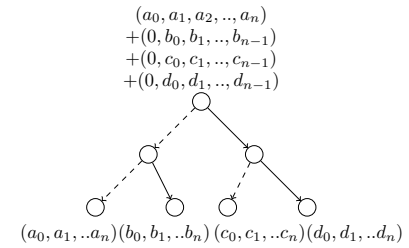

Fig. 5: Computing weight enumeration for one node.

The complexity of computing the weight enumeration of a given code represented as a BDD is $\mathcal{O}(N)$, where N is the number of nodes in the BDD and adding two integer vectors counts as a unit operation. In terms of single integer additions, the complexity is $\mathcal{O}(nN)$.

We have described the algorithm for computing the weight enumeration when the code represented as a BDD is regarded as being over $GF(4)$. Going back to the general case of an $[n, k]$ linear code over $GF(2)$, we can easily modify the algorithm to compute the weight distribution for any binary linear code when it is represented as paths in a BDD.

Computing the weight distribution in general is a hard problem, that can only be solved by brute force. The naive way of doing it (without the BDD representation) is to run through all the code words and count their weights. This has complexity $\mathcal{O}(2^k)$. If $k > n/2$, the complexity of doing weight enumeration becomes bigger than it needs to be. Then one can compute the weight distribution for the dual code (of dimension $n-k$ and complexity $\mathcal{O}(2^{n-k}) < \mathcal{O}(2^k)$), and use MacWilliams' identity [11, p. 127] to find the weight distribution of the given code. Theorem 4 gives a unified approach to calculate the weight enumeration irrespective of whether k is bigger or smaller than $n/2$.

4 Classification for Rankwidth 1 graphs

The algorithm for classifying self-dual codes corresponding to general graphs as described in [8]: Let L_{n-1} be the set of representatives for classes of graphs on $n-1$ vertices corresponding to equivalent self-dual codes.

- Compute the set of graphs E_n by adding a vertex to each graph in L_{n-1} in $2^{n-1}-1$ ways i.e. making the vertex adjacent to every possible non-empty subset of the vertex set.
- For each set of of isomorphic graphs keep only one graph in E_n.
- Use weight-enumerators to partition the set E_n i.e. graphs corresponding to same weight-enumerators are put in one class.
- Partition each class in E_n by checking for self-dual equivalence.
- Output L_n that contains one graph from each class in E_n.

We utilize the following definition of distance hereditary graphs.

Theorem 5 ([2]). *Let G be a finite graph with at least two vertices. Then G is distance-hereditary if and only if G is obtained from an edge by a sequence of one of the following vertex extensions: add vertex as a pendant, add vertex as a true-twin to an existing vertex and add vertex as a false-twin to an existing vertex.*

Let $\mathcal{G}_{n-1}$ be all connected graphs of rankwidth 1 on $n-1$ vertices. Then $\mathcal{G}_n$ can be obtained by adding a vertex to each graph in $\mathcal{G}_{n-1}$ as a pendant or a twin to some vertex. Consider $\mathcal{C}$ to be the orbit of a graph $G \in \mathcal{G}_{n-1}$. Let $G_1, G_2 \in \mathcal{C}$. Then there is a sequence of LC operations S that can take G_1 to G_2. Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be the $3(n-1)$ extensions of G_1 and G_2 obtained via adding pendants or twins. We show that via applying S on any graph in $\mathcal{E}_1$, we end-up with a graph in $\mathcal{E}_2$ implying that $\mathcal{E}_1$ and $\mathcal{E}_2$ are *LC-equivalent*.

Let u be a new vertex added to G_1 as a pendant or twin to a vertex $v \in V(G_1)$. The LC operations at vertices in G_1 switch the role of u relative to v as a pendant or a twin. At the same time, G_1 changes to G_2 after S has been performed. Then, u can be seen as being attached to G_2 as a pendant or twin (according to what happens after apply S to $G_1 + u$). Hence, any graph $\mathcal{E}_2$ can be seen as being obtained from a graph in $\mathcal{E}_1$ via applying S. This implies that instead of considering extensions of $\mathcal{C}$, we need only consider extensions of just one representative from $\mathcal{C}$. Let L_{n-1} be the set of representatives of all orbits in $\mathcal{G}_{n-1}$. Since rankwidth is preserved by LC operations, the graphs in the sets L_{n-1}, E_n and L_n must be of rankwidth 1. Hence by above discussion, in the computation of E_n from L_{n-1}, the vertex must be added as a pendant or a false-twin or a true-twin. There are at most $3(n-1)$ ways to do that. So instead of branching in $2^{n-1}-1$ ways, we need only branch in at most $3n-3$ ways. Furthermore, the isomorphism testing in E_n is linear in n for rankwidth 1 graphs.

5 Minimum Distance

Glynn et al [13] showed that the minimum distance of a code is equal to one plus the minimum vertex degree over all graphs in the corresponding LC orbit.

Lemma 3. *If a connected graph contains a twin-pair, then the minimum distance of the corresponding code is 2.*

Lemma 4. *Codes with corresponding graphs of rankwidth 1 have minimum distance 2.*

Lemma 5. *If a graph contains a twin-pair, then every graph in its LC orbit will contain a twin-pair or a pendant.*

Lemma 6. *If G does not have a pendant or a twin-pair, then no graph in the LC orbit of G will have a twin-pair.*

Combining Lemma 3, Lemma 5 and Lemma 6 gives the following theorem.

Theorem 6. *The minimum distance of a self-dual additive code over $GF(4)$ is at least 3 if and only if the corresponding graph G has no pendants or twin-pairs.*

5.1 An approximation algorithm for minimum distance

The problem of computing the minimum distance of a binary linear code is NP-hard [32]. In addition, the problem is hard to approximate within any constant factor in random polynomial time [10]. For self-dual codes over $GF(4)$, the minimum distance is $1 + \delta$ where δ is the minimum degree of any vertex in any graph in the LC orbit of the graph corresponding to the code. It is possible to get the minimum distance from the weight enumerator polynomial or from the LC orbit, but both these approaches take exponential time. In this section we discuss a heuristic approach to get some upper bound on the minimum distance.

Computing δ is equivalent to finding a sequence of LC operations starting at some vertex u such that at the end, there is a vertex of degree δ in the resulting graph. Clearly, finding this sequence is hard. The strategy we use is to pick a vertex in the graph and try to decrease its degree as much as possible via LC operations.

The sequence of LC operations corresponds to a path in a Breadth-First-Search (BFS) tree T of G with u as the root vertex. A BFS tree is constructed as follows: Pick u as root. At each layer i, the vertices in layer i are neighbors of vertices in layer $i-1$ that have not been already placed in some layer.

We aim to use this tree to find a path that gives a sequence of LC operations to decrease the degree of u. If no such path exists, then the algorithm reports the degree of u as a candidate for δ. See Figure 6. Now, we state the algorithm:

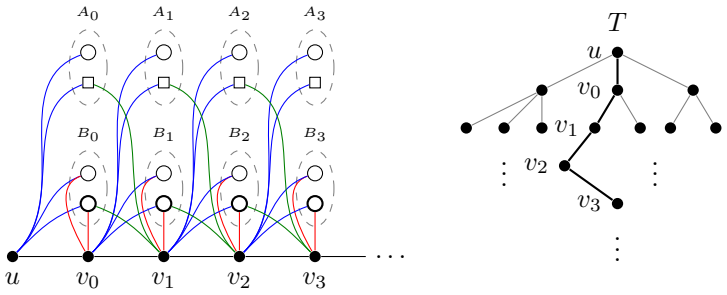

Fig. 6: Decomposition of the graph G along a path $uv_0v_1v_2v_3\ldots$ in the BFS-tree T with u as root vertex. Black-filled circles represent vertices. Other shapes represent sets. An edge from a vertex to a set represents that every vertex in the set is a neighbor of the vertex. Edges between sets have not been shown.

1: Construct BFS tree For $u \in V(G)$, construct Breadth-First-Search tree T with u as the root node. The neighborhood of a vertex at layer i in T lie only in layers $i-1, i$ and $i+1$. Note that we would require to reconstruct the tree after LC operations along the path. The tree T is used to guide the sequence of LC operations to be applied to decrease the degree of u.

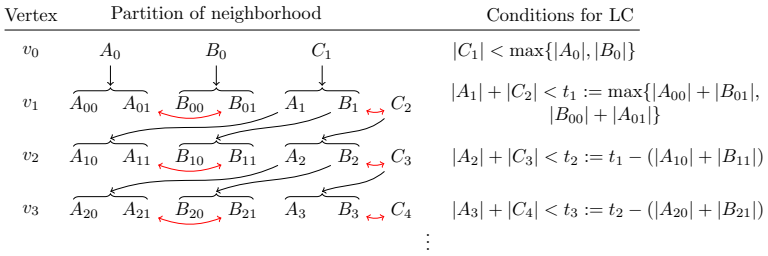

Fig. 7: At each vertex v_i store sets A_i, B_i, C_i and t_i. If $|A_3|+|C_4| < t_3$, then $|A_2|+|C_3| < t_2$ which in turn implies $|A_1| + |C_2| < t_1$. Hence, just by looking at v_3 we can decide whether degree of u can be decreased.

2: Partition neighborhoods At each node of the tree, we store some information that can be used to check whether LC along the path to the root will decrease the degree of u.
At the root node u, we have $C = N(u)$.
At the second layer in the tree, for each vertex $v_0 \in C_0 = N(u)$, the neighborhood of v_0 can be partitioned as (apart from u) as (A_0, B_0, C_1) where $B_0 = C_0 \cap N(v_0), A_0 = C_0 - B_0$ and $C_1 = N(v_0) - (u \cup C_0)$.
At the ith layer with $j = i-1, k = i+1$, for each vertex $v_i \in C_i$, partition the neighborhood of v_i as (apart from v_j) $(A_{j0}, A_{j1}, B_{j0}, B_{j1}, A_i, B_i, C_k)$ where $A_{j0} = A_j - N(v_i), A_{j1} = A_j - A_{j0}, B_{j0} = B_j - N(v_i), B_{j1} = B_j - B_{j0}, B_i = C_i - N(v_i), A_i = C_i - B_i, C_k = N(v_i) - (v_j \cup A_{j1} \cup B_{j1} \cup B_i)$. See Figure 6.

3: Book keeping For each $i \geq 2$ with $j = i-1, k = i+1$, at each vertex v_i, we store the sets A_i, B_i, C_k and the values $a_i := |A_{j0}| + |B_{j1}|, t_i := t_j - a_i$ and $|A_i| + |C_k|$.

4: Check for LC If $|A_i|+|C_k| \leq t_i$, then an LC operation along the path from v_i to the root will decrease the degree of u. See Figure 7. Then apply LC operations along this path and construct the BFS-tree T for the new graph and repeat.

5: Return degree of u If there does not exist any vertex in the tree with $|A_i|+|C_k| \leq t_i$, return the current degree of u as δ_u.

6: Terminate Finally, the algorithm outputs the smallest δ_u over all vertices in the graph.

Running Time: For a given graph G, the BFS-tree can be constructed in linear time. For each vertex v, the partition for neighborhood of v can be computed in polynomial time and it will take polynomial space to store the necessary information. Hence, in polynomial time we can decide whether there exists a path in the tree along which LC operations decreases the degree of u. Hence, the algorithm terminates in polynomial time.

References

[1] Christine Bachoc and Philippe Gaborit. "On Extremal Additive I_4 Codes of Length 10 to 18". In: *Electronic Notes in Discrete Mathematics* 6 (2001), pp. 55–64. DOI: 10.1016/S1571-0653(04)00157-X. URL: https://doi.org/10.1016/S1571-0653(04)00157-X.

[2] Hans-Jürgen Bandelt and Henry Martyn Mulder. "Distance-hereditary graphs". In: *J. Comb. Theory, Ser. B* 41.2 (1986), pp. 182–208. DOI: 10.1016/0095-8956(86)90043-2. URL: https://doi.org/10.1016/0095-8956(86)90043-2.

[3] André Bouchet. "Graphic presentations of isotropic systems". In: *Journal of Combinatorial Theory, Series B* 45.1 (1988), pp. 58 –76. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(88)90055-X. URL: http://www.sciencedirect.com/science/article/pii/009589568890055X.

[4] Stefka Bouyuklieva and Zlatko Varbanov. "Some connections between self-dual codes, combinatorial designs and secret sharing schemes". In: 5 (May 2011).

[5] A. Robert Calderbank et al. "Quantum Error Correction Via Codes Over GF(4)". In: *IEEE Trans. Information Theory* 44.4 (1998), pp. 1369–1387. DOI: 10.1109/18.681315. URL: https://doi.org/10.1109/18.681315.

[6] Lars Eirik Danielsen and Matthew G. Parker. "Edge Local Complementation and Equivalence of Binary Linear Codes". In: *CoRR* abs/0710.2243 (2007). arXiv: 0710.2243. URL: http://arxiv.org/abs/0710.2243.

[7] Lars Eirik Danielsen and Matthew G. Parker. "Edge local complementation and equivalence of binary linear codes". In: *Des. Codes Cryptography* 49.1-3 (2008), pp. 161–170. DOI: 10.1007/s10623-008-9190-x. URL: https://doi.org/10.1007/s10623-008-9190-x.

[8] Lars Eirik Danielsen and Matthew G. Parker. "On the classification of all self-dual additive codes over GF(4) of length up to 12". In: *J. Comb. Theory, Ser. A* 113.7 (2006), pp. 1351–1367. DOI: 10.1016/j.jcta.2005.12.004. URL: https://doi.org/10.1016/j.jcta.2005.12.004.

[9] Lars Eirik Danielsen and Matthew G. Parker. "Spectral Orbits and Peak-to-Average Power Ratio of Boolean Functions with Respect to the $\{I, H, N\}^n$ Transform". In: *Sequences and Their Applications - SETA 2004, Third International Conference, Seoul, Korea, October 24-28, 2004, Revised Selected Papers*. 2004, pp. 373–388. DOI: 10.1007/11423461_28. URL: https://doi.org/10.1007/11423461_28.

[10] Ilya Dumer, Daniele Micciancio, and Madhu Sudan. "Hardness of approximating the minimum distance of a linear code". In: *IEEE Trans. Information Theory* 49.1 (2003), pp. 22–37. DOI: 10.1109/TIT.2002.806118. URL: https://doi.org/10.1109/TIT.2002.806118.

[11] F.J.Macwilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[12] Philippe Gaborit et al. "On additive $GF(4)$ codes". In: *Codes and Association Schemes, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 9-12, 1999*. 1999, pp. 135–150.

[13] David Glynn et al. *The geometry of additive quantum codes*. Jan. 2004.
[14] Daniel Gottesman. "Stabilizer Codes ad Quantum Error Correction". In: *Phd Thesis, Caltech* (May 1997). DOI: arXiv:quant-ph/9705052.
[15] Martin Grohe and Pascal Schweitzer. "Isomorphism Testing for Graphs of Bounded Rank Width". In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. 2015, pp. 1010–1029. DOI: 10.1109/FOCS.2015.66. URL: https://doi.org/10.1109/FOCS.2015.66.
[16] T. Aaron Gulliver and Masaaki Harada. "Certain Self-Dual Codes over Z_4 and the Odd Leech Lattice". In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 12th International Symposium, AAECC-12, Toulouse, France, June 23-27, 1997, Proceedings*. 1997, pp. 130–137. DOI: 10.1007/3-540-63163-1_10. URL: https://doi.org/10.1007/3-540-63163-1_10.
[17] Masaaki Harada. "Optimal self-dual Z4-codes and a unimodular lattice in dimension 41". In: *Finite Fields and Their Applications* 18.3 (2012), pp. 529 –536. ISSN: 1071-5797. DOI: https://doi.org/10.1016/j.ffa.2011.11.004. URL: http://www.sciencedirect.com/science/article/pii/S1071579711001018.
[18] M. Hein, J. Eisert, and H. J. Briegel. "Multiparty entanglement in graph states". In: *Phys. Rev. A* 69 (6 2004), p. 062311. DOI: 10.1103/PhysRevA.69.062311. URL: https://link.aps.org/doi/10.1103/PhysRevA.69.062311.
[19] Jon-Lark Kim and Nari Lee. "Secret sharing schemes based on additive codes over GF(4)". In: *Appl. Algebra Eng. Commun. Comput.* 28.1 (2017), pp. 79–97. DOI: 10.1007/s00200-016-0296-5. URL: https://doi.org/10.1007/s00200-016-0296-5.
[20] Shin-ichi Minato. "πDD: A New Decision Diagram for Efficient Problem Solving in Permutation Space". In: *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*. 2011, pp. 90–104. DOI: 10.1007/978-3-642-21581-0_9. URL: https://doi.org/10.1007/978-3-642-21581-0_9.
[21] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. "Graphical description of the action of local Clifford transformations on graph states". In: *Phys. Rev. A* 69 (2 2004), p. 022316. DOI: 10.1103/PhysRevA.69.022316. URL: https://link.aps.org/doi/10.1103/PhysRevA.69.022316.
[22] Sang-il Oum. "Rank-width and vertex-minors". In: *J. Comb. Theory, Ser. B* 95.1 (2005), pp. 79–100. DOI: 10.1016/j.jctb.2005.03.003. URL: https://doi.org/10.1016/j.jctb.2005.03.003.
[23] Vera Pless and N.J.A Sloane. "On the classification and enumeration of self-dual codes". In: *Journal of Combinatorial Theory, Series A* 18.3 (1975), pp. 313 –335. ISSN: 0097-3165. DOI: https://doi.org/10.1016/0097-

3165(75)90042-4. URL: http://www.sciencedirect.com/science/article/pii/0097316575900424.

[24] Håvard Raddum and Srimathi Varadharajan. "Factorization Using Binary Decision Diagrams". In: *Cryptography and Communications* (2018). to appear.

[25] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. "Measurement-based quantum computation on cluster states". In: *Phys. Rev. A* 68 (2 2003), p. 022312. DOI: 10.1103/PhysRevA.68.022312. URL: https://link.aps.org/doi/10.1103/PhysRevA.68.022312.

[26] Richard Rudell. "Dynamic variable ordering for ordered binary decision diagrams". In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993*. 1993, pp. 42–47. DOI: 10.1109/ICCAD.1993.580029. URL: https://doi.org/10.1109/ICCAD.1993.580029.

[27] Thorsten Ernst Schilling and Håvard Raddum. "Solving Compressed Right Hand Side Equation Systems with Linear Absorption". In: *Sequences and Their Applications - SETA 2012 - 7th International Conference, Waterloo, ON, Canada, June 4-8, 2012. Proceedings*. 2012, pp. 291–302. DOI: 10.1007/978-3-642-30615-0_27. URL: https://doi.org/10.1007/978-3-642-30615-0_27.

[28] D. Schlingemann. "Stabilizer Codes Can Be Realized As Graph Codes". In: *Quantum Info. Comput.* 2.4 (June 2002), pp. 307–323. ISSN: 1533-7146. URL: http://dl.acm.org/citation.cfm?id=2011477.2011481.

[29] S Mesnager ST Dougherty and P Solé. "Secret-sharing schemes based on self-dual codes". In: *Information Theory Workshop, Porto* (2008), pp. 338–342.

[30] Ryuhei Uehara and Takeaki Uno. *Canonical tree representation of distance hereditary graphs and its applications*. Tech. rep. 2006.

[31] M Van Den Nest. "Local Equivalence of Stabilizer States and Codes". In: *Phd thesis, K. U. Leuven, Belgium* (May 2005).

[32] Alexander Vardy. "The intractability of computing the minimum distance of a code". In: *IEEE Trans. Information Theory* 43.6 (1997), pp. 1757–1766. DOI: 10.1109/18.641542. URL: https://doi.org/10.1109/18.641542.