Space-Optimal Quasi-Gray Codes with Logarithmic Read Complexity

Diptarka Chakraborty
Computer Science Institute of Charles University, Prague, Czech Republic
diptarka@iuuk.mff.cuni.cz

Debarati Das
Computer Science Institute of Charles University, Prague, Czech Republic
debaratix710@gmail.com

Michal Koucký
Computer Science Institute of Charles University, Prague, Czech Republic
koucky@iuuk.mff.cuni.cz

Nitin Saurabh
Max-Planck-Institut für Informatik, Saarbrücken, Germany
nsaurabh@mpi-inf.mpg.de

Abstract

A quasi-Gray code of dimension n and length ℓ over an alphabet Σ is a sequence of distinct words $w_1, w_2, \ldots, w_\ell$ from Σ^n such that any two consecutive words differ in at most c coordinates, for some fixed constant $c > 0$. In this paper we are interested in the read and write complexity of quasi-Gray codes in the bit-probe model, where we measure the number of symbols read and written in order to transform any word w_i into its successor w_{i+1}.

We present construction of quasi-Gray codes of dimension n and length 3^n over the ternary alphabet $\{0, 1, 2\}$ with worst-case read complexity $O(\log n)$ and write complexity 2. This generalizes to arbitrary odd-size alphabets. For the binary alphabet, we present quasi-Gray codes of dimension n and length at least $2^n - 20n$ with worst-case read complexity $6 + \log n$ and write complexity 2. This complements a recent result by Raskin [Raskin '17] who shows that any quasi-Gray code over binary alphabet of length 2^n has read complexity $\Omega(n)$.

Our results significantly improve on previously known constructions and for the odd-size alphabets we break the $\Omega(n)$ worst-case barrier for space-optimal (non-redundant) quasi-Gray codes with constant number of writes. We obtain our results via a novel application of algebraic tools together with the principles of catalytic computation [Buhrman et al. '14, Ben-Or and Cleve '92, Barrington '89, Coppersmith and Grossman '75].

2012 ACM Subject Classification Theory of computation → Cell probe models and lower bounds

Keywords and phrases Gray code, Space-optimal counter, Decision assignment tree, Cell probe model

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.12

Related Version [8], https://arxiv.org/abs/1712.01834

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787. The third author was also partially supported by the Center of Excellence CE-ITI under the grant P202/12/G061 of GA ČR.

26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 12; pp. 12:1–12:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements Authors would like to thank Gerth Stølting Brodal for bringing the problem to our attention, and to Petr Gregor for giving a talk on space-optimal counters in our seminar, which motivated this research. Authors thank Meena Mahajan and Venkatesh Raman for pointing out the result in [39]. We also thank anonymous reviewers for helpful suggestions that improved the presentation of the paper.

1 Introduction

One of the fundamental problems in the domain of algorithm design is to list down all the objects belonging to a certain combinatorial class. Researchers are interested in efficient generation of a list such that an element in the list can be obtained by a small amount of change to the element that precedes it. One of the classic examples is the binary *Gray code* introduced by Gray [23], initially used in pulse code communication. The original idea of a Gray code was to list down all binary strings of length n, i.e, all the elements of $\mathbb{Z}_2^n$, such that any two successive strings differ by exactly one bit. The idea was later generalized for other combinatorial classes (e.g. see [37, 28]). Gray codes have found applications in a wide variety of areas, such as information storage and retrieval [9], processor allocation [10], computing the permanent [37], circuit testing [41], data compression [40], graphics and image processing [1], signal encoding [32], modulation schemes for flash memories [26, 22, 44] and many more. Interested reader may refer to an excellent survey by Savage [42] for a comprehensive treatment on this subject.

In this paper we study the construction of Gray codes over $\mathbb{Z}_m^n$ for any $m \in \mathbb{N}$. Originally, Gray codes were meant to list down all the elements from its domain but later studies (e.g. [20, 38, 5, 6]) focused on the generalization where we list ℓ distinct elements from the domain, each two consecutive elements differing in one position. We refer to such codes as Gray codes of length ℓ [20]. When the code lists all the elements from its domain it is referred to as *space-optimal*. It is often required that the last and the first strings appearing in the list also differ in one position. Such codes are called *cyclic Gray codes*. Throughout this paper we consider only cyclic Gray codes and we refer to them simply as Gray codes. Researchers also study codes where two successive strings differ in at most c positions, for some fixed constant $c > 0$, instead of differing in exactly one position. Such codes are called *quasi-Gray codes* [5][1] or c-Gray codes.

We study the problem of constructing quasi-Gray codes over $\mathbb{Z}_m^n$ in the cell probe model [43], where each cell stores an element from $\mathbb{Z}_m$. The efficiency of a construction is measured using three parameters. First, we want the length of a quasi-Gray code to be as large as possible. Ideally, we want space-optimal codes. Second, we want to minimize the number of coordinates of the input string the algorithm reads in order to generate the next (or, previous) string in the code. Finally, we also want the number of cells written in order to generate the successor (or, predecessor) string to be as small as possible. Since our focus is on quasi-Gray codes, the number of writes will always be bounded by a universal constant. We are interested in the worst-case behavior and we use *decision assignment trees* (DAT) of Fredman [20] to measure these complexities.

The second requirement of the above is motivated from the study of *loopless generation* of combinatorial objects. In the loopless generation we are required to generate the next string from the code in constant time. Different loopless algorithms to generate Gray codes

[1] Readers may note that the definition of quasi-Gray code given in [20] was different. The code referred as quasi-Gray code by Fredman [20] is called Gray code of length ℓ where $\ell < m^n$, in our notation.

are known in the literature [17, 4, 28]. However, those algorithms use extra memory cells in addition to the space required for the input string which makes it impossible to get a space-optimal code from them. More specifically, our goal is to design a decision assignment tree on n variables to generate a code over the domain $\mathbb{Z}_m^n$. If we allow extra memory cells (as in the case of loopless algorithms) then the corresponding DAT will be on $n+b$ variables, where b is the number of extra memory cells used.

Although there are known quasi-Gray codes with logarithmic read complexity and constant write complexity [20, 38, 5, 6], none of these constructions is space-optimal. The best result misses at least 2^{n-t} strings from the domain when having read complexity $t+O(\log n)$ [6]. Despite of an extensive research under many names, e.g., construction of Gray codes [20, 36, 16, 24], dynamic language membership problem [19], efficient representation of integers [38, 6], so far we do not have any quasi-Gray code of length $2^n - 2^{\epsilon n}$, for some constant $\epsilon < 1$, with worst-case read complexity $o(n)$ and write complexity $o(n)$. The best worst-case read complexity for *space-optimal* Gray code is $n-1$ [21]. Recently, Raskin [39] showed that any space-optimal quasi-Gray code over the domain $\mathbb{Z}_2^n$ must have read complexity $\Omega(n)$. This lower bound is true even if we allow non-constant write complexity. It is worth noting that this result can be extended to the domain $\mathbb{Z}_m^n$ when m is even.

In this paper we show that such lower bound does not hold for quasi-Gray codes over $\mathbb{Z}_m^n$, when m is odd. In particular, we construct space-optimal quasi-Gray codes over $\{0,1,2\}^n$ with read complexity $4\log_3 n$ and write complexity 2. As a consequence we get an exponential separation between the read complexity of space-optimal quasi-Gray code over $\mathbb{Z}_2^n$ and that over $\mathbb{Z}_3^n$.

▶ **Theorem 1.** *Let $m \in \mathbb{N}$ be odd and $n \in \mathbb{N}$ be such that $n \geq 15$. Then, there is a space-optimal quasi-Gray code C over $\mathbb{Z}_m^n$ for which, the two functions $next(C,w)$ and $prev(C,w)$ can be implemented by inspecting at most $4\log_m n$ cells while writing only 2 cells.*

In the statement of the above theorem, $next(C,w)$ denotes the element appearing after w in the cyclic sequence of the code C, and analogously, $prev(C,w)$ denotes the preceding element. Using the argument as in [20, 36] it is easy to see a lower bound of $\Omega(\log_m n)$ on the read complexity when the domain is $\mathbb{Z}_m^n$. Hence our result is optimal up to some small constant factor.

Raskin shows $\Omega(n)$ lower bound on the read complexity of space-optimal binary quasi-Gray codes. The existence of binary quasi-Gray codes with sub-linear read complexity of length $2^n - 2^{\epsilon n}$, for some constant $\epsilon < 1$, was open. Using a different technique than that used in the proof of Theorem 1 we get a quasi-Gray code over the binary alphabet which enumerates all but $O(n)$ many strings. This result generalizes to the domain $\mathbb{Z}_q^n$ for any prime power q.

▶ **Theorem 2.** *Let $n \geq 15$ be any natural number. Then, there is a quasi-Gray code C of length at least $2^n - 20n$ over $\mathbb{Z}_2^n$, such that the two functions $next(C,w)$ and $prev(C,w)$ can be implemented by inspecting at most $6+\log n$ cells while writing only 2 cells.*

We remark that the points that are missing from C in the above theorem are all of the form $\{0,1\}^{O(\log n)}0^{n-O(\log n)}$.

If we are allowed to read and write constant fraction of n bits then Theorem 2 can be adapted to get a quasi-Gray code of length $2^n - O(1)$ (see Section 5). In this way we get a trade-off between length of the quasi-Gray code and the number of bits read in the worst-case. All of our constructions can be made uniform.

Using the Chinese Remainder Theorem (cf. [14]), we also develop a technique that allows us to compose Gray codes over various domains. Hence, from quasi-Gray codes over domains

$\mathbb{Z}^n_{m_1}, \mathbb{Z}^n_{m_2}, \ldots, \mathbb{Z}^n_{m_k}$, where m_i's are pairwise co-prime, we can construct quasi-Gray codes over $\mathbb{Z}^{n'}_m$, where $m = m_1 \cdot m_2 \cdots m_k$. Using this technique on our main results, we get a quasi-Gray code over $\mathbb{Z}^n_m$ for *any* $m \in \mathbb{N}$ that misses only $O(n \cdot o^n)$ strings where $m = 2^\ell o$ for an odd o, while achieving the read complexity similar to that stated in Theorem 1. It is worth mentioning that if we get a space-optimal quasi-Gray code over the binary alphabet with non-trivial savings in read complexity, then we will have a space-optimal quasi-Gray code over the strings of alphabet $\mathbb{Z}_m$ for any $m \in \mathbb{N}$ with similar savings.

The technique by which we construct our quasi-Gray codes relies heavily on simple algebra which is a substantial departure from previous mostly combinatorial constructions. We view Gray codes as permutations on $\mathbb{Z}^n_m$ and we decompose them into k simpler permutations on $\mathbb{Z}^n_m$, each being computable with read complexity 3 and write complexity 1. Then we apply a different composition theorem, than mentioned above, to obtain space-optimal quasi-Gray codes on $\mathbb{Z}^{n'}_m$, $n' = n + \log k$, with read complexity $O(1) + \log k$ and write complexity 2. The main issue is the decomposition of permutations into few simple permutations. This is achieved by techniques of catalytic computation [7] going back to the work of Coppersmith and Grossman [13, 2, 3].

It follows from the work of Coppersmith and Grossman [13] that our technique is incapable of designing a space-optimal quasi-Gray code on $\mathbb{Z}^{n'}_2$ as any such code represents an *odd* permutation. The tools we use give inherently only *even* permutations. However, we can construct quasi-Gray codes from cycles of length $2^n - 1$ on $\mathbb{Z}^n_2$ as they are even permutations. Indeed, that is what we do for our Theorem 2. We note that any efficiently computable odd permutation on $\mathbb{Z}^n_2$, with say read complexity $(1 - \epsilon)n$ and write complexity $O(1)$, could be used together with our technique to construct a space-optimal quasi-Gray code on $\mathbb{Z}^{n'}_2$ with read complexity at most $(1 - \epsilon')n'$ and constant write complexity. This would represent a major progress on space-optimal Gray codes. (We would compose the odd permutation with some even permutation to obtain a full cycle on $\mathbb{Z}^n_2$. The size of the decomposition of the even permutation into simpler permutations would govern the read complexity of the resulting quasi-Gray code.)

Interestingly, Raskin's result relies on showing that a decision assignment tree of sub-linear read complexity must compute an even permutation.

1.1 Related works

The construction of Gray codes is central to the design of algorithms for many combinatorial problems [42]. Frank Gray [23] first came up with a construction of Gray code over binary strings of length n, where to generate the successor or predecessor strings one needs to read n bits in the worst-case. The type of code described in [23] is known as *binary reflected Gray code*. Later Bose *et al.* [5] provided a different type of Gray code construction, namely *recursive partition Gray code* which attains $O(\log n)$ average case read complexity while having the same worst-case read requirements. The read complexity we referred here is in the bit-probe model. It is easy to observe that any space-optimal binary Gray code must read $\log n + 1$ bits in the worst-case [20, 36, 21]. Recently, this lower bound was improved to $n/2$ in [39]. An upper bound of even $n - 1$ was not known until very recently [21]. This is also the best known so far.

Fredman [20] extended the definition of Gray codes by considering codes that may not enumerate all the strings (though presented in a slightly different way in [20]) and also introduced the notion of *decision assignment tree* (DAT) to study the complexity of any code in the bit-probe model. He provided a construction that generates a Gray code of length $2^{c \cdot n}$ for some constant $c < 1$ while reducing the worst-case bit-read to $O(\log n)$. Using

Table 1 Taxonomy of construction of Gray/quasi-Gray codes over $\mathbb{Z}_m^n$.

Reference	Value of m	length	Worst-case cell read	Worst-case cell write
[23]	2	2^n	n	1
[20]	2	$2^{\Theta(n)}$	$O(\log n)$	$O(1)$
[19]	2	$\Theta(2^n/n)$	$\log n + 1$	$\log n + 1$
[38]	2	2^{n-1}	$\log n + 4$	4
[5]	2	$2^n - O(2^n/n^t)$	$O(t \log n)$	3
[6]	2	$2^n - 2^{n-t}$	$\log n + t + 3$	2
[6]	2	$2^n - 2^{n-t}$	$\log n + t + 2$	3
[21]	2	2^n	$n - 1$	1
Theorem 2	2	$2^n - O(n)$	$\log n + 4$	2
[12]	any m	m^n	n	1
Theorem 1	any odd m	m^n	$4 \log_m n + 3$	2

the idea of Lucal's modified reflected binary code [31], Munro and Rahman [38] got a code of length 2^{n-1} with worst-case read complexity only $4 + \log n$. However in their code two successive strings differ by 4 coordinates in the worst-case, instead of just one and we refer to such codes as quasi-Gray codes following the nomenclature used in [5]. Brodal et al. [6] extended the results of [38] by constructing a quasi-Gray code of length $2^n - 2^{n-t}$ for arbitrary $1 \leq t \leq n - \log n - 1$, that has $t + 3 + \log n$ bits ($t + 2 + \log n$ bits) worst-case read complexity and any two successive strings in the code differ by at most 2 bits (3 bits).

In contrast to the Gray codes over binary alphabets, Gray codes over non-binary alphabets received much less attention. The construction of binary reflected Gray code was generalized to the alphabet $\mathbb{Z}_m$ for any $m \in \mathbb{N}$ in [18, 12, 27, 40, 28, 25]. However, each of those constructions reads n coordinates in the worst-case to generate the next element. As mentioned before, we measure the read complexity in the well studied cell probe model [43] where we assume that each cell stores an element of $\mathbb{Z}_m$. The argument of Fredman in [20] implies a lower bound of $\Omega(\log_m n)$ on the read complexity of quasi-Gray code on $\mathbb{Z}_m^n$. To the best of our knowledge, for non-binary alphabets, there is nothing known similar to the results of Munro and Rahman or Brodal et al. [38, 6]. We summarize the previous results along with ours in Table 1.

Additionally, many variants of Gray codes have been studied in the literature. A particular one that has garnered a lot of attention in the past 30 years is the well-known *middle levels conjecture*. See [33, 34, 35, 24], and the references therein. It has been established only recently [33]. The conjecture says that there exists a Hamiltonian cycle in the graph induced by the vertices on levels n and $n + 1$ of the hypercube graph in $2n + 1$ dimensions. In other words, there exists a Gray code on the middle levels. Mütze et al. [34, 35] studied the question of efficiently enumerating such a Gray code in the word RAM model. They [35] gave an algorithm to enumerate a Gray code in the middle levels that requires $O(n)$ space and *on average* takes $O(1)$ time to generate the next vertex. In this paper we consider the bit-probe model, and Gray codes over the complete hypercube. It would be interesting to know whether our technique can be applied for the middle level Gray codes.

1.2 Our technique

Our construction of Gray codes relies heavily on the notion of s-functions defined by Coppersmith and Grossman [13]. An s-function is a permutation τ on $\mathbb{Z}_m^n$ defined by a function $f : \mathbb{Z}_m^s \to \mathbb{Z}_m$ and an $(s + 1)$-tuple of indices $i_1, i_2, \ldots, i_s, j \in [n]$ such that

$\tau(\langle x_1, x_2, \ldots, x_n\rangle) = (\langle x_1, x_2, \ldots, x_{j-1}, x_j + f(x_{i_1}, \ldots, x_{i_s}), x_{j+1}, \ldots, x_n\rangle)$, where the addition is inside $\mathbb{Z}_m$. Each s-function can be computed by some decision assignment tree that given a vector $x = \langle x_1, x_2, \ldots, x_n\rangle$, inspects $s+1$ coordinates of x and then it writes into a single coordinate of x.

A counter C (quasi-Gray code) on $\mathbb{Z}_m^n$ can be thought of as a permutation on $\mathbb{Z}_m^n$. Our goal is to construct some permutation α on $\mathbb{Z}_m^n$ that can be written as a composition of 2-functions $\alpha_1, \ldots, \alpha_k$, i.e., $\alpha = \alpha_k \circ \alpha_{k-1} \circ \cdots \circ \alpha_1$.

Given such a decomposition, we can build another counter C' on $\mathbb{Z}_m^{r+n}$, where $r = \lceil \log_m k \rceil$, for which the function $next(C', x)$ operates as follows. The first r-coordinates of x serve as an *instruction pointer* $i \in [m^r]$ that determines which α_i should be executed on the remaining n coordinates of x. Hence, based on the current value i of the r coordinates, we perform α_i on the remaining coordinates and then we update the value of i to $i+1$. (For $i > k$ we can execute the identity permutation which does nothing.)

We can use known Gray codes on $\mathbb{Z}_m^r$ to represent the instruction pointer so that when incrementing i we only need to write into one of the coordinates. This gives a counter C' which can be computed by a decision assignment tree that reads $r+3$ coordinates and writes into 2 coordinates of x. (A similar composition technique is implicit in Brodal et al. [6].) If C is of length $\ell = m^n - t$, then C' is of length $m^{n+r} - tm^r$. In particular, if C is space-optimal then so is C'.

Hence, we reduce the problem of constructing 2-Gray codes to the problem of designing large cycles in $\mathbb{Z}_m^n$ that can be decomposed into 2-functions. Coppersmith and Grossman [13] studied precisely the question of, which permutations on $\mathbb{Z}_2^n$ can be written as a composition of 2-functions. They show that a permutation on $\mathbb{Z}_2^n$ can be written as a composition of 2-functions if and only if the permutation is even. Since $\mathbb{Z}_2^n$ is of even size, a cycle of length 2^n on $\mathbb{Z}_2^n$ is an odd permutation and thus it cannot be represented as a composition of 2-functions. However, their result also implies that a cycle of length $2^n - 1$ on $\mathbb{Z}_2^n$ *can* be decomposed into 2-functions.

We want to use the counter composition technique described above in connection with a cycle of length $2^n - 1$. To maximize the length of the cycle C' in $\mathbb{Z}_2^{n+r}$, we need to minimize k, the number of 2-functions in the decomposition. By a simple counting argument, most cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ require k to be exponentially large in n. This is too large for our purposes. Luckily, there are cycles of length $2^n - 1$ on $\mathbb{Z}_2^n$ that can be decomposed into polynomially many 2-functions, and we obtain such cycles from linear transformations.

There are linear transformations $\mathbb{Z}_2^n \to \mathbb{Z}_2^n$ which define a cycle on $\mathbb{Z}_2^n$ of length $2^n - 1$. For example, the matrix corresponding to the multiplication by a fixed generator of the multiplicative group $\mathbb{F}_{2^n}^*$ of the Galois field $GF[2^n]$ is such a matrix. Such matrices are full rank and they can be decomposed into $O(n^2)$ elementary matrices, each corresponding to a 2-function. Moreover, there are matrices derived from primitive polynomials that can be decomposed into at most $4n$ elementary matrices.[2] We use them to get a counter on $\mathbb{Z}_2^{n'}$ of length at least $2^{n'} - 20n'$ whose successor and predecessor functions are computable by decision assignment trees of read complexity $\leq 6 + \log n'$ and write complexity 2. Such counter represents 2-Gray code of the prescribed length. For any prime q, the same construction yields 2-Gray codes of length at least $q^{n'} - 5q^2n'$ with decision assignment trees of read complexity $\leq 6 + \log_q n'$ and write complexity 2.

[2] Primitive polynomials were previously also used in a similar problem, namely to construct shift-register sequences (see e.g. [28]).

The results of Coppersmith and Grossman [13] can be generalized to $\mathbb{Z}_m^n$ as stated in Richard Cleve's thesis [11].[3] For odd m, if a permutation on $\mathbb{Z}_m^n$ is even then it can be decomposed into 2-functions. Since m^n is odd, a cycle of length m^n on $\mathbb{Z}_m^n$ is an even permutation and so it can be decomposed into 2-functions. If the number k of those functions is small, so the $\log_m k$ is small, we get the sought after counter with small read complexity. However, for most cycles of length m^n on $\mathbb{Z}_m^n$, k is exponential in n.

We show though, that there is a cycle α of length m^n on $\mathbb{Z}_m^n$ that can be decomposed into $O(n^3)$ 2-functions. This in turn gives space-optimal 2-Gray codes on $\mathbb{Z}_m^{n'}$ with decision assignment trees of read complexity $O(\log_m n')$ and write complexity 2.

We obtain the cycle α and its decomposition in two steps. First, for $i \in [n]$, we consider the permutation α_i on $\mathbb{Z}_m^n$ which maps each element $0^{i-1}ay$ onto $0^{i-1}(a+1)y$, for $a \in \mathbb{Z}_m$ and $y \in \mathbb{Z}_m^{n-i}$, while other elements are mapped to themselves. Hence, α_i is a product of m^{n-i} disjoint cycles of length m. We show that $\alpha = \alpha_n \circ \alpha_{n-1} \circ \cdots \circ \alpha_1$ is a cycle of length m^n. In the next step we decompose each α_i into $O(n^2)$ 2-functions.

For $i \leq n-2$, we can decompose α_i using the technique of Ben-Or and Cleve [3] and its refinement in the form of catalytic computation of Buhrman et al. [7]. We can think of $x \in \mathbb{Z}_m^n$ as content of n memory registers, where $x_1, \ldots, x_{i-1}$ are the input registers, x_i is the output register, and $x_{i+1}, \ldots, x_n$ are the working registers. The catalytic computation technique gives a program consisting of $O(n^2)$ instructions, each being equivalent to a 2-function, which performs the desired adjustment of x_i based on the values of $x_1, \ldots, x_{i-1}$ without changing the ultimate values of the other registers. (We need to increment x_i iff $x_1, \ldots, x_{i-1}$ are all zero.) This program directly gives the desired decomposition of α_i, for $i \leq n-2$. (Our proof in Section 6 uses the language of permutations.)

The technique of catalytic computation fails for α_{n-1} and α_n as the program needs at least two working registers to operate. Hence, for α_{n-1} and α_n we have to develop entirely different technique. This is not trivial and quite technical but it is nevertheless possible, thanks to the specific structure of α_{n-1} and α_n.

2 Preliminaries

In the rest of the paper we only present constructions of the successor function $next(C, w)$ for our codes. Since all the operations in those constructions are readily invertible, the same arguments also give the predecessor function $prev(C, w)$.

Notation: We use the standard notions of groups and fields, and mostly we will use only elementary facts about them (see [15, 30] for background.). By $\mathbb{Z}_m$ we mean the set of integers modulo m, i.e., $\mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z}$. Throughout this paper whenever we use addition and multiplication operation between two elements of $\mathbb{Z}_m$, then we mean the operations within $\mathbb{Z}_m$ that is modulo m. For any $m \in \mathbb{N}$, we let $[m]$ denote the set $\{1, 2, \ldots, m\}$. Unless stated otherwise explicitly, all the logarithms we consider throughout this paper are based 2.

Now we define the notion of counters used in this paper.

▶ **Definition 3** (Counter). A *counter* of length ℓ over a domain $\mathcal{D}$ is any cyclic sequence $C = (w_1, \ldots, w_\ell)$ such that $w_1, \ldots, w_\ell$ are distinct elements of $\mathcal{D}$. With the counter C we associate two functions $next(C, w)$ and $prev(C, w)$ that give the successor and predecessor element of w in C, that is for $i \in [\ell]$, $next(C, w_i) = w_j$ where $j - i = 1 \bmod \ell$, and $prev(C, w_i) = w_k$ where $i - k = 1 \bmod \ell$. If $\ell = |\mathcal{D}|$, we call the counter a *space-optimal counter*.

[3] Unfortunately, there is no written record of the proof.

Often elements in the underlying domain $\mathcal{D}$ have some "structure" to them. In such cases, it is desirable to have a counter such that consecutive elements in the sequence differ by a "small" change in the "structure". We make this concrete in the following definition.

▶ **Definition 4** (Gray Code)**.** Let $\mathcal{D}_1, \ldots, \mathcal{D}_n$ be finite sets. A *Gray code* of length ℓ over the domain $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ is a counter $(w_1, \ldots, w_\ell)$ of length ℓ over $\mathcal{D}$ such that any two consecutive strings w_i and w_j, $j - i = 1 \bmod \ell$, differ in exactly one coordinate when viewed as an n-tuple. More generally, if for some constant $c \geq 1$, any two consecutive strings w_i and w_j, $j - i = 1 \bmod \ell$, differ in at most c coordinates such a counter is called a *c-Gray Code*.

By a quasi-Gray code we mean c-Gray code for some unspecified fixed $c > 0$. In the literature sometimes people do not place any restriction on the relationship between w_ℓ and w_1 and they refer to such a sequence a (quasi)-Gray code. In their terms, our codes would be *cyclic* (quasi)-Gray codes. If $\ell = |\mathcal{D}|$, we call the codes *space-optimal* (quasi-)Gray codes.

Decision Assignment Tree: The computational model we consider in this paper is called *Decision Assignment Tree* (DAT). The definition we provide below is a generalization of that given in [20]. It is intended to capture random access machines with small word size.

Let us fix an underlying domain $\mathcal{D}^n$ whose elements we wish to enumerate. In the following, we will denote an element in $\mathcal{D}^n$ by $\langle x_1, x_2, \ldots, x_n \rangle$. A decision assignment tree is a $|\mathcal{D}|$-ary tree such that each internal node is labeled by one of the variables $x_1, x_2, \ldots, x_n$. Furthermore, each outgoing edge of an internal node is labeled with a distinct element of $\mathcal{D}$. Each leaf node of the tree is labeled by a set of assignment instructions that set new (fixed) values to chosen variables. The variables which are not mentioned in the assignment instructions remain unchanged.

The execution on a decision assignment tree on a particular input vector $\langle x_1, \ldots, x_n \rangle \in \mathcal{D}^n$ starts from the root of the tree and continues in the following way: at a non-leaf node labeled with a variable x_i, the execution queries x_i and depending on the value of x_i the control passes to the node following the outgoing edge labeled with the value of x_i. Upon reaching a leaf, the corresponding set of assignment statements is used to modify the vector $\langle x_1, x_2, \ldots, x_n \rangle$ and the execution terminates. The modified vector is the output of the execution.

Thus, each decision assignment tree computes a mapping from $\mathcal{D}^n$ into $\mathcal{D}^n$. We are interested in decision assignment trees computing the mapping $next(C, \langle x_1, x_2, \ldots, x_n \rangle)$ for some counter C. When C is space-optimal we can assume, without loss of generality, that each leaf assigns values only to the variables that it reads on the path from the root to the leaf. (Otherwise, the decision assignment tree does not compute a bijection.) We define the *read complexity* of a decision assignment tree T, denoted by $\mathrm{READ}(T)$, as the maximum number of non-leaf nodes along any path from the root to a leaf. Observe that any mapping from $\mathcal{D}^n$ into $\mathcal{D}^n$ can be implemented by a decision assignment tree with read complexity n. We also define the *write complexity* of a decision assignment tree T, denoted by $\mathrm{WRITE}(T)$, as the maximum number of assignment instructions in any leaf.

Instead of the domain $\mathcal{D}^n$, we will sometimes also use domains that are a cartesian product of different domains. The definition of a decision assignment tree naturally extends to this case of different variables having different domains.

For any counter $C = (w_1, \ldots, w_\ell)$, we say that C is computed by a decision assignment tree T if and only if for $i \in [\ell]$, $next(C, w_i) = T(w_i)$, where $T(w_i)$ denotes the output string obtained after an execution of T on w_i. Note that any two consecutive strings in the cyclic sequence of C differ by at most $\mathrm{WRITE}(T)$ many coordinates. For a small constant $c \geq 1$, some domain $\mathcal{D}$, and all large enough n, we will be interested in constructing cyclic counters on $\mathcal{D}^n$ that are computed by decision assignment trees of write complexity c and read complexity $O(\log n)$. By definition such cyclic counters will necessarily be c-Gray codes.

2.1 Construction of Gray codes

For our construction of quasi-Gray codes on a domain $\mathcal{D}^n$ with decision assignment trees of small read and write complexity we will need ordinary Gray codes on a domain $\mathcal{D}^{O(\log n)}$. Several constructions of space-optimal binary Gray codes are known where the oldest one is the binary reflected Gray code [23]. This can be generalized to space-optimal (cyclic) Gray codes over non-binary alphabets (see e.g. [12, 28]).

▶ **Theorem 5** ([12, 28]). *For any $m, n \in \mathbb{N}$, there is a space-optimal (cyclic) Gray code over $\mathbb{Z}_m^n$.*

3 Chinese Remainder Theorem for Counters

Below we describe how to compose decision assignment trees over different domains to get a decision assignment tree for a larger mixed domain. For all the details and proofs we refer the reader to the full version of this paper [8].

▶ **Theorem 6** (Chinese Remainder Theorem for Counters). *Let $r, n_1, \ldots, n_r \in \mathbb{N}$ be integers, and let $\mathcal{D}_{1,1}, \ldots, \mathcal{D}_{1,n_1}, \mathcal{D}_{2,1}, \ldots, \mathcal{D}_{r,n_r}$ be some finite sets of size at least two. Let $\ell_1 \geq r-1$ be an integer, and $\ell_2, \ldots, \ell_r$ be pairwise co-prime integers. For $i \in [r]$, let C_i be a counter of length ℓ_i over $\mathcal{D}_i = \mathcal{D}_{i,1} \times \cdots \times \mathcal{D}_{i,n_i}$ computed by a decision assignment tree T_i over n_i variables. Then, there exists a decision assignment tree T over $\sum_{i=1}^r n_i$ variables that implements a counter C of length $\prod_{i=1}^r \ell_i$ over $\mathcal{D}_1 \times \cdots \times \mathcal{D}_r$. Furthermore, $\mathrm{READ}(T) = n_1 + \max\{\mathrm{READ}(T_i)\}_{i=2}^r$, and $\mathrm{WRITE}(T) = \mathrm{WRITE}(T_1) + \max\{\mathrm{WRITE}(T_i)\}_{i=2}^r$.*

We remark that if C_i's in the theorem are space-optimal then so is C. The proof of the theorem constructs a special type of a counter where we always read the first coordinate, increment it, and further depending on its value, we may update the value of another coordinate. Note, for such type of a counter the co-primality condition is necessary at least for $\ell_1 = 2, 3$ (see the full version [8]).

As a corollary of the above theorem, to get a decision assignment tree implementing space-optimal quasi-Gray codes over $\mathbb{Z}_m$ for any $m \in \mathbb{N}$, we only need decision assignment trees implementing space-optimal quasi-Gray codes over $\mathbb{Z}_2$ and $\mathbb{Z}_m$, for odd m.

4 Permutation Group and Construction of Counters

We start this section with some basic notation and facts about the permutation group which we will use heavily in the rest of the paper. The set of all permutations over a domain $\mathcal{D}$ forms a group under the composition operation, denoted by $\circ$, which is defined as follows: for any two permutations σ and α, $\sigma \circ \alpha(x) = \sigma(\alpha(x))$, where $x \in \mathcal{D}$. The corresponding group, denoted $\mathcal{S}_N$, is the *symmetric group* of order $N = |\mathcal{D}|$. We say, a permutation $\sigma \in \mathcal{S}_N$ is a *cycle* of length ℓ if there are distinct elements $a_1, \ldots, a_\ell \in [N]$ such that for $i \in [\ell - 1]$, $a_{i+1} = \sigma(a_i)$, $a_1 = \sigma(a_\ell)$, and for all $a \in [N] \setminus \{a_1, a_2, \ldots, a_\ell\}$, $\sigma(a) = a$. We denote such a cycle by $(a_1, a_2, \cdots, a_\ell)$.

Roughly speaking, a counter of length ℓ over $\mathcal{D}$, in the language of permutations, is nothing but a cycle of the same length in $\mathcal{S}_{|\mathcal{D}|}$. We now make this correspondence precise and give a construction of a decision assignment tree that implements such a counter.

We state our key lemma to construct Gray codes from a decomposition of a permutation.

▶ **Lemma 7.** *Let $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_r$ be a domain. Suppose $\sigma_1, \ldots, \sigma_k \in \mathcal{S}_{|\mathcal{D}|}$ are such that $\sigma = \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$ is a cycle of length ℓ. Let $T_1, \ldots, T_k$ be decision assignment trees that implement $\sigma_1, \ldots, \sigma_k$ respectively. Let $\mathcal{D}' = \mathcal{D}'_1 \times \cdots \times \mathcal{D}'_{r'}$ be a domain such that $|\mathcal{D}'| \geq k$,*

and let T' be a decision assignment tree that implements a counter C' of length k' over $\mathcal{D}'$ where $k' \geq k$.

Then, there exists a decision assignment tree T that implements a counter of length $k'\ell$ over $\mathcal{D}' \times \mathcal{D}$ such that $\mathrm{READ}(T) = r' + \max\{\mathrm{READ}(T_i)\}_{i=1}^{k}$, *and* $\mathrm{WRITE}(T) = \mathrm{WRITE}(T') + \max\{\mathrm{WRITE}(T_i)\}_{i=1}^{k}$.

Proof sketch. Suppose $C' = (a_1, \ldots, a_{k'})$. Now let us consider the following procedure P: on any input $\langle x_1, x_2 \rangle \in \mathcal{D}' \times \mathcal{D}$, if $x_1 = a_j$ for some $j \in [k]$, set $x_2 \leftarrow \sigma_j(x_2)$. Next, increment the first coordinate, i.e., set $x_1 \leftarrow next(C', x_1)$.

Read and write complexity of the statement follows immediately. The correctness also follows from some basic property of permutation group together with the fact that σ is a cycle of length ℓ. ◀

In the next two sections we describe the construction of $\sigma_1, \cdots, \sigma_k \in \mathcal{S}_N$ where $N = m^n$ for some $m, n \in \mathbb{N}$ and how the value of k depends on the length of the cycle $\sigma = \sigma_k \circ \sigma_{k-1} \circ \cdots \circ \sigma_1$.

5 Counters via Linear Transformation

The construction in this section is based on linear transformations. Consider the vector space $\mathbb{F}_q^n$, and let $L : \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a linear transformation. A basic fact in linear algebra says that if L has *full* rank, then the mapping given by L is a bijection. Thus, when L is full rank, the mapping can also be thought of as a permutation over $\mathbb{F}_q^n$. Throughout this section we use many basic terms related to linear transformation without defining them, for the details of which we refer the reader to any standard text book on linear algebra (e.g. [29]).

A natural way to build counter out of a full rank linear transformation is to fix a starting element, and repeatedly apply the linear transformation to obtain the next element (cf. [28]). Clearly this only list out elements in the cycle containing the starting element. Therefore, we would like to choose the starting element such that we enumerate the largest cycle. Ideally, we would like the largest cycle to contain all the elements of $\mathbb{F}_q^n$. However this is not possible because any linear transformation fixes the all-zero vector. But there do exist full rank linear transformations such that the permutation given by them is a single cycle of length $q^n - 1$. Such a linear transformation would give us a counter over a domain of size q^n that enumerates all but one element. Clearly, a trivial implementation of the aforementioned argument would lead to a counter that reads and writes all n coordinates in the worst-case. In the rest of this section, we will develop an implementation and argue about the choice of linear transformation such that the read and write complexity decreases exponentially.

It is well known that every linear transformation L is associated with some matrix $A \in \mathbb{F}_q^{n \times n}$ such that applying the linear transformation is equivalent to the left multiplication by A. Furthermore, L has full rank iff A is invertible over $\mathbb{F}_q$.

▶ **Definition 8** (Elementary matrices). An $n \times n$ matrix over a field $\mathbb{F}$ is said to be an *elementary matrix* if it has one of the following forms:

(a) The off-diagonal entries are all 0. For some $i \in [n]$, (i, i)-th entry is a non-zero $c \in \mathbb{F}$. Rest of the diagonal entries are 1.

(b) The diagonal entries are all 1. For some i and j, $1 \leq i \neq j \leq n$, (i, j)-th entry is a non-zero $c \in \mathbb{F}$. Rest of the off-diagonal entries are 0.

From the definition it is easy to see that left multiplication by an elementary matrix of the first type is equivalent to multiplying the i-th row with c, and by an elementary matrix of the second type it is equivalent to adding c times j-th row to the i-th row.

▶ **Proposition 9.** *Let $A \in \mathbb{F}^{n\times n}$ be invertible. Then A can be written as a product of k elementary matrices such that $k \leq n^2 + 4(n-1)$.*

The proof follows from Gaussian elimination.

5.1 Construction of the counter

Let A be a full rank linear transformation from $\mathbb{F}_q^n$ to $\mathbb{F}_q^n$ such that when viewed as permutation it is a single cycle of length $q^n - 1$. More specifically, A is an invertible matrix in $\mathbb{F}_q^{n\times n}$ such that for any $x \in \mathbb{F}_q^n$ where $x \neq (0,\ldots,0)$, $Ax, A^2x, \ldots, A^{(q^n-1)}x$ are distinct. Such a matrix exists, for example, take A to be the matrix of a linear transformation that corresponds to multiplication from left by a fixed generator of the multiplicative group of $\mathbb{F}_{q^n}$ under the standard vector representation of elements of $\mathbb{F}_{q^n}$. Let $A = E_k E_{k-1} \cdots E_1$ where E_i's are elementary matrices.

▶ **Theorem 10.** *Let q, A, and k be as defined above. Let $r \geq \log_q k$. There exists a quasi-Gray code on the domain $(\mathbb{F}_q)^{n+r}$ of length $q^{n+r} - q^r$ that can be implemented using a decision assignment tree T such that* $\mathrm{READ}(T) \leq r + 2$ *and* $\mathrm{WRITE}(T) \leq 2$.

Proof. The proof follows readily from Lemma 7, where E_i's play the role of σ_i's, and noting that the permutation given by any elementary matrix can be implemented using a decision assignment tree that reads at most two coordinates and writes at most one. For the counter C' on $(\mathbb{F}_q)^r$ we chose a Gray code of trivial read complexity r and write complexity 1. ◀

Thus, we obtain a counter on a domain of size roughly kq^n that misses at most qk elements. Clearly, we would like to minimize k. A trivial bound on k is $O(n^2)$ that follows from Proposition 9. We now discuss the choice of A so that k becomes $O(n)$ based on primitive polynomials over finite fields.

Let $p(z)$ be a primitive polynomial of degree n over $\mathbb{F}_q$, where $p(z) = z^n + c_{n-1}z^{n-1} + c_{n-2}z^{n-2} + \cdots + c_1 z + c_0$. The matrix A defined as follows is the matrix representing multiplication by some generator (a root of $p(z)$) of the multiplicative group of $\mathbb{F}_{q^n}$:

$$\begin{pmatrix} -c_{n-1} & 1 & 0 & \cdots & 0 \\ -c_{n-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -c_1 & 0 & 0 & \cdots & 1 \\ -c_0 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

It is easy to see that A can be written as a product of at most $n + 4(n-1)$ elementary matrices. (In case, q is a power of 2, then the number of elementary matrices in the product is at most $n + 3(n-1)$.) Hence, from the discussion above and using Theorem 10, we obtain the following corollaries. Setting $r = \lceil \log(4n-3) \rceil$ in Theorem 10 gives:

▶ **Corollary 11.** *For any $n' \geq 2$, and $n = n' + \lceil \log(4n'-3) \rceil$, there exists a counter on $(\mathbb{Z}_2)^n$ that misses at most $8n$ strings and can be implemented by a decision assignment tree that reads at most $4 + \log n$ bits and writes at most 2 bits.*

By doubling the number of missed strings and increasing the number of read bits by one we can construct given counters for any $\mathbb{Z}_2^n$, where $n \geq 15$. For the general case, when q is a prime power, we obtain the following corollary by setting r to $\lceil \log_q(5n-4) \rceil$ or $1 + \lceil \log_q(5n-4) \rceil$ in Theorem 10.

▶ **Corollary 12** (Generalization of Theorem 2). *Let q be any prime power. For $n \geq 15$, there exists a counter on $\mathbb{Z}_q^n$ that misses at most $5q^2n$ strings and that is computed by a decision assignment tree with read complexity at most $6 + \log_q n$ and write complexity 2.*

6 Space-optimal Counters over $\mathbb{Z}_m^n$ for any Odd m

In this section we sketch a proof of Theorem 1. We want to use Lemma 7. Set $n' = n - c \cdot \log n$ for a suitable constant $c > 0$. We define permutations $\alpha_1, \cdots, \alpha_{n'} \in \mathcal{S}_N$, for $N = m^{n'}$, such that $\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$. We will show that each of these α_i's can be further decomposed into $\alpha_{i,1}, \cdots, \alpha_{i,j} \in \mathcal{S}_N$ for some j, such that each of $\alpha_{i,r}$ for $r \in [j]$ can be implemented using DAT with read complexity 3 and write complexity 1. Finally we use Lemma 7 by considering all these $\alpha_{i,r}$'s as $\sigma_1, \cdots, \sigma_k$, where k is $O(mn'^3)$.

For any $i \in [n']$, define α_i as follows: for any $\langle x_1, \cdots, x_{n'} \rangle \in \mathbb{Z}_m^{n'}$, if $x_j = 0$ for all $j = 1, \cdots, i-1$, then $x_i \leftarrow x_i + 1 \mod m$. Observe, $\alpha = \alpha_{n'} \circ \cdots \circ \alpha_1$ is a cycle of length $m^{n'}$. Notice each α_i is a $(i-1)$-function on $\mathbb{Z}_m^{n'}$ (see Section 1.2 for their definition). Now if for any $i \in [n']$ we can find a set of 2-functions $\alpha_{i,1}, \cdots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$, then we can use them as σ_j's in Lemma 7. As a result the complexity in Theorem 1 follows.

Note α_1, α_2 and α_3 are already 2-functions. In the case of α_i for $4 \leq i \leq n' - 2$, we can directly adopt the technique from [3, 7] to generate the desired set of 2-functions. For $i = n' - 1$, it is possible to generalize the proof technique of [13] to decompose $\alpha_{n'-1}$ but we have to develop a new technique to decompose $\alpha_{n'}$.

▶ **Lemma 13.** *For any $4 \leq i \leq n' - 2$, one can construct a set of 2-functions $\alpha_{i,1}, \cdots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$ where $k_i \leq c_1 \cdot (i-1)^2$ for some constant $c_1 > 0$.*

It remains to decompose $\alpha_{n'-1}$ and $\alpha_{n'}$. A key tool is the following lemma.

▶ **Lemma 14.** *Suppose there are two cycles, $\sigma = (t, a_1, \cdots, a_{\ell-1})$ and $\tau = (t, b_1, \cdots, b_{\ell-1})$, of length $\ell \geq 2$ such that $a_i \neq b_j$ for every $i, j \in [\ell - 1]$. Then, $(\sigma \circ \tau)^\ell \circ (\tau \circ \sigma)^\ell = \sigma^2$.*

Let us now consider $\alpha_{n'}$, the case of $\alpha_{n'-1}$ is analogous. For $a = (m+1)/2$, we define $\sigma = (\langle 00 \cdots 0(0 \cdot a) \rangle, \langle 00 \cdots 0(1 \cdot a) \rangle, \cdots, \langle 00 \cdots 0((m-1) \cdot a) \rangle)$, and $\tau = (\langle (0 \cdot a) 00 \cdots 0 \rangle, \langle (1 \cdot a) 00 \cdots 0 \rangle, \cdots, \langle ((m-1) \cdot a) 00 \cdots 0 \rangle)$, where the multiplication is in $\mathbb{Z}_m$. In other words, we define σ by adding a to the value of the last coordinate when all other coordinates are set to 0, and we define τ by adding a to the value of the first coordinate when all other coordinates are set to 0. Since m is co-prime with $(m+1)/2$, σ and τ are cycles of length m. (Here we use the fact that m is odd.) Observe that $\sigma^2 = \alpha_{n'}$, so by applying Lemma 14 to σ and τ we get $\alpha_{n'}$. It might seem we didn't make much progress towards decomposition, as now instead of one $(n'-1)$-function $\alpha_{n'}$ we have to decompose two $(n'-1)$-functions σ and τ. However, we will not decompose σ and τ directly, but rather we obtain a decomposition for $(\sigma \circ \tau)^m$ and $(\tau \circ \sigma)^m$. Surprisingly this can be done using a generalization of Lemma 13.

We consider an $(n'-3)$-function σ' whose cycle decomposition contains σ as one of its cycles. Similarly we consider a 3-function τ' whose cycle decomposition contains τ as one of its cycles. We carefully choose these σ' and τ' such that $(\sigma' \circ \tau')^m = (\sigma \circ \tau)^m$ and $(\tau' \circ \sigma')^m = (\tau \circ \sigma)^m$. We will decompose σ' and τ' to get the desired decomposition.

▶ **Lemma 15.** *For any $i \in \{n'-1, n'\}$, one can construct a set of 2-functions $\alpha_{i,1}, \cdots, \alpha_{i,k_i}$ such that $\alpha_i = \alpha_{i,k_i} \circ \cdots \circ \alpha_{i,1}$ where $k_i \leq c_2 \cdot m \cdot (i-1)^2$ for some constant $c_2 > 0$.*

References

1. D. J. Amalraj, N. Sundararajan, and Goutam Dhar. Data structure based on Gray code encoding for graphics and image processing. In *Proceedings of the SPIE: International Society for Optical Engineering*, pages 65–76, 1990.
2. David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.
3. Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992. doi:10.1137/0221006.
4. James R. Bitner, Gideon Ehrlich, and Edward M. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Commun. ACM*, 19(9):517–521, 1976.
5. Prosenjit Bose, Paz Carmi, Dana Jansens, Anil Maheshwari, Pat Morin, and Michiel H. M. Smid. Improved methods for generating quasi-Gray codes. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 224–235, 2010. doi:10.1007/978-3-642-13731-0_22.
6. Gerth Stølting Brodal, Mark Greve, Vineet Pandey, and Srinivasa Rao Satti. Integer representations towards efficient counting in the bit probe model. *J. Discrete Algorithms*, 26:34–44, 2014. doi:10.1016/j.jda.2013.11.001.
7. Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 857–866, 2014. doi:10.1145/2591796.2591874.
8. Diptarka Chakraborty, Debarati Das, Michal Koucký, and Nitin Saurabh. Optimal quasi-Gray codes: Does the alphabet matter? *CoRR*, 2017. arXiv:1712.01834.
9. C. C. Chang, H. Y. Chen, and C. Y. Chen. Symbolic Gray code as a data allocation scheme for two-disc systems. *The Computer Journal*, 35(3):299–305, 1992. doi:10.1093/comjnl/35.3.299.
10. M. S. Chen and K. G. Shin. Subcube allocation and task migration in hypercube multiprocessors. *IEEE Transactions on Computers*, 39(9):1146–1155, 1990. doi:10.1109/12.57056.
11. Richard Cleve. *Methodologies for Designing Block Ciphers and Cryptographic Protocols*. PhD thesis, University of Toronto, April 1989.
12. Martin Cohn. Affine m-ary Gray codes. *Information and Control*, 6(1):70–78, 1963.
13. Don Coppersmith and Edna Grossman. Generators for certain alternating groups with applications to cryptography. *SIAM J. Appl. Math.*, 29(4):624–627, 1975.
14. C. Ding, D. Pei, and A. Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.
15. David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley & Sons, 2004.
16. Tomáš Dvořák, Petr Gregor, and Václav Koubek. Generalized Gray codes with prescribed ends. *Theor. Comput. Sci.*, 668:70–94, 2017. doi:10.1016/j.tcs.2017.01.010.
17. Gideon Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. ACM*, 20(3):500–513, 1973. doi:10.1145/321765.321781.
18. Ivan Flores. Reflected number systems. *IRE Transactions on Electronic Computers*, EC-5(2):79–82, 1956.
19. Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.
20. Michael L. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM J. Comput.*, 7(2):134–146, 1978. doi:10.1137/0207012.

21 Zachary Frenette. Towards the efficient generation of Gray codes in the bitprobe model. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2016.

22 E. Gad, M. Langberg, M. Schwartz, and J. Bruck. Constant-weight Gray codes for local rank modulation. *IEEE Transactions on Information Theory*, 57(11):7431–7442, 2011. doi:10.1109/TIT.2011.2162570.

23 F. Gray. Pulse code communication, 1953. US Patent 2,632,058. URL: http://www.google.com/patents/US2632058.

24 Petr Gregor and Torsten Mütze. Trimming and gluing Gray codes. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 40:1–40:14, 2017. doi:10.4230/LIPIcs.STACS.2017.40.

25 Felix Herter and Günter Rote. Loopless Gray code enumeration and the tower of bucharest. In *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*, pages 19:1–19:19, 2016.

26 A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *IEEE Transactions on Information Theory*, 55(6):2659–2673, 2009. doi:10.1109/TIT.2009.2018336.

27 James T. Joichi, Dennis E. White, and S. G. Williamson. Combinatorial Gray codes. *SIAM J. Comput.*, 9(1):130–141, 1980. doi:10.1137/0209013.

28 Donald E. Knuth. *The Art of Computer Programming. Volume 4A: Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, 2011.

29 Serge Lang. *Linear Algebra.* Undergraduate Texts in Mathematics. Springer New York, 1987.

30 Rudolf Lidl and Harald Niederreiter. *Finite Fields.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 1996. doi:10.1017/CBO9780511525926.

31 H. M. Lucal. Arithmetic operations for digital computers using a modified reflected binary code. *IRE Transactions on Electronic Computers*, EC-8(4):449–458, 1959.

32 J. Ludman. Gray code generation for mpsk signals. *IEEE Transactions on Communications*, 29(10):1519–1522, 1981. doi:10.1109/TCOM.1981.1094886.

33 Torsten Mütze. Proof of the middle levels conjecture. *Proceedings of the London Mathematical Society*, 112(4):677–713, 2016.

34 Torsten Mütze and Jerri Nummenpalo. Efficient computation of middle levels Gray codes. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 915–927, 2015.

35 Torsten Mütze and Jerri Nummenpalo. A constant-time algorithm for middle levels Gray codes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2238–2253, 2017.

36 Patrick K. Nicholson, Venkatesh Raman, and S. Srinivasa Rao. A survey of data structures in the bitprobe model. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 303–318, 2013. doi:10.1007/978-3-642-40273-9_19.

37 Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms.* Academic Press, 1978.

38 M. Ziaur Rahman and J. Ian Munro. Integer representation and counting in the bit probe model. *Algorithmica*, 56(1):105–127, 2010. doi:10.1007/s00453-008-9247-2.

39 Mikhail Raskin. A linear lower bound for incrementing a space-optimal integer representation in the bit-probe model. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 88:1–88:12, 2017.

40 Dana Richards. Data compression and Gray-code sorting. *Information Processing Letters*, 22(4):201–205, 1986. doi:10.1016/0020-0190(86)90029-3.

41 J. P. Robinson and M. Cohn. Counting sequences. *IEEE Transactions on Computers*, C-30(1):17–23, 1981. doi:10.1109/TC.1981.6312153.

42 Carla Savage. A survey of combinatorial Gray codes. *SIAM review*, 39(4):605–629, 1997.

43 Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981. doi:10.1145/322261.322274.

44 Y. Yehezkeally and M. Schwartz. Snake-in-the-box codes for rank modulation. *IEEE Transactions on Information Theory*, 58(8):5471–5483, 2012. doi:10.1109/TIT.2012.2196755.