UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Teele Tars

Asymptotic Bounds on the Length of Functional Batch and PIR Codes

Bachelor's Thesis (9 ECTS)

Supervisor: Vitaly Skachek, PhD

Tartu 2025

Asymptotic Bounds on the Length of Functional Batch and PIR Codes

Abstract: Batch codes were first introduced in 2004 by Ishai et al. to be used in distributed storage systems for load balancing. Batch codes can be viewed as a special case of another family of codes called Private Information Retrieval (PIR) codes. PIR codes are used for retrieving information from a distributed storage system so that the system does not know what information was requested. This thesis studies variants of batch and PIR codes called functional batch and functional PIR codes, respectively. For both of these types of codes, it is desirable to find codes that can support as many simultaneous queries as possible, while minimizing the storage overhead. Bounds on the length of these codes for a fixed query size are presented and a simplex-based near-optimal construction for these bounds is described. The bounds are compared with the minimum code lengths found by computer search.

Keywords: Coding theory, batch codes, Private Information Retrieval (PIR) codes

CERCS: P170, Computer science, numerical analysis, systems, control

Funktsionaalsete partiikoodide ja privaatse infootsingu koodide pikkuse asümptootilised tõkked

Lühikokkuvõte: Partiikoode tutvustasid esimesena aastal 2004 Ishai jt hajusfailisüsteemide koormusjaotuse tasakaalustamise eesmärgil. Partiikoode võib vaadelda ühe teise koodiliigi, privaatse infootsingu (PIO) koodide erijuhuna. PIO-koode kasutatakse selleks, et pärida süsteemilt infot, ilma et süsteem päringu sisu kohta midagi teada saaks. Töös on käsitletud partii- ja PIO-koodide variante, mida nimetatakse vastavalt funktsionaalseteks partiikoodideks ja funktsionaalseteks PIO-koodideks. Mõlema kooditüübi puhul on eesmärgiks leida koode, mis võimaldavad korraga vastata võimalikult paljudele päringutele, hoides talletatud andmete hulga võimalikult väiksena. Töös on esitatud tõkked selliste koodide pikkuse kohta, kui päringu suurus on fikseeritud, ja kirjeldatud on ka simpleks-koodidel põhinevat konstruktsiooni, mis on esitatud tõkete suhtes peaaegu optimaalne. Tõkkeid võrreldakse arvutiotsingu abil leitud vähima koodipikkuse väärtustega.

Võtmesõnad: Kodeerimisteooria, partiikoodid, privaatse infootsingu (PIO) koodid

CERCS: P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

This thesis studies two families of codes: functional batch codes and functional PIR codes.

Batch codes were first introduced by Ishai et al. [1] for the purpose of load balancing in distributed data storage systems. In a distributed data storage system, the original data is stored in a database. This data is then replicated and distributed across multiple nodes (also called servers or buckets). When a user requests an element from the original database, it is retrieved using elements from the distributed nodes. The system should be able to simultaneously retrieve as many requested elements as necessary, while reducing the storage overhead. Batch codes are used to describe such distribution schemes. The parameters of the code describe the characteristics of the system, such as the size of the original database, the number of nodes and the size of the largest recoverable batch of elements. A batch code request consists of a batch (of a certain size) of the original database elements. Functional batch codes are a generalization of batch codes, where the requests consist of a batch of linear combinations of the original database elements.

Private Information Retrieval (PIR) describes a scenario where the user wants to retrieve information from a database without the database gaining any knowledge of the information that was requested. A naive approach to this would be to request all database elements at once. However, this approach is not suitable for large databases due to the large amount of data downloaded to retrieve just one element. Also, this solution can only ensure computational privacy (i.e. the privacy depends on the computational limitations of the system). Distributed storage systems offer information-theoretic privacy, which means that privacy is ensured even if the computing power of the system is unbounded (assuming that the servers do not collude). A typical PIR scheme is similar to that of the batch codes, except that a PIR request consists of copies of one original database element (in other words, the requested elements are identical). Functional PIR codes are a generalization of PIR codes, where a request consists of identical copies of some linear combination of original database elements.

In the case of both (functional) batch and (functional) PIR codes, it is desirable to find the minimal amount of storage needed in order to allow for retrieving user requests (also called the code length), given the size of the original database and the size of a request. If the request size is fixed, there exists some relation between the size of the original database and the length of the code. The thesis presents asymptotic results on the minimum value of the code length, as the size of the original database approaches infinity. Section 3 gives definitions for the terms used in the thesis. Four bounds are presented in Section 4, a lower and an upper bound for functional batch codes and related bounds for functional PIR codes. Finally, experimental results for small parameter values are compared with the bounds in Section 5.

2 Overview of Literature and Related Works

Batch and PIR codes have been studied extensively after their introduction. Several constructions of batch codes with a near-optimal redundancy for a fixed batch size and large code dimension were presented in a paper by Vardy and Yaakobi [2], achieving redundancy $O(\sqrt{k}\log k)$, where k is the dimension of the code. This result was improved upon (along with other results) in another paper by Asi and Yaakobi [3], where the authors presented constructions of PIR and batch codes with a fixed request size, as well as several asymptotic bounds on the redundancy achieved by those constructions. A survey of batch and PIR codes by Skachek [4] presents various bounds of the parameters of PIR and batch codes, and discusses relations between batch and PIR codes and locally repairable codes (LRC).

The study of linear batch codes (which are studied in this thesis also) was initiated in a paper by Lipmaa and Skachek [5]. In that paper, linear batch codes were related to classical error-correcting codes and bounds were derived based on this relation. Constructions of larger linear batch codes from the existing smaller codes were also presented. Another family of codes related to batch codes, called switch codes, were studied in a paper by Wang et al. [6]. The authors presented a construction of binary switch codes based on simplex codes, which is analogous to a construction used in this thesis, only here it is used for functional batch codes.

Functional variants of batch and PIR codes have also been studied in various works. In a paper by Zhang et al. [7], various bounds on the length of functional PIR and batch codes were presented. This included two bounds for which the alternative proofs in this thesis are given. In the paper, it was also proved that the binary simplex code is a 2^{k-1}-functional PIR code and conjectured that it is also a 2^{k-1}-functional batch code. The gap between this conjecture and existing knowledge was narrowed in a paper by Yohananov and Yaakobi [8], where it was shown that for the request size $t = \left\lfloor \frac{5}{6}2^{k-1} \right\rfloor - k$, the bound $n \leq 2^k - 1$ holds, having used a construction based on extended simplex codes to achieve this result. A paper by Hollmann et al. [9] studied the batch code properties of the simplex code. In the paper, it was proved that the binary simplex code of dimension k and length $n = 2^k - 1$ can serve any request of length 2^{k-1} consisting of odd-weight vectors. This knowledge is applied in the constructions used in this thesis. The authors also presented a stronger conjecture than the previously mentioned conjecture that the binary simplex code is a 2^{k-1}-functional batch code, stating that the recovery sets for the functional batch requests are of size at most two.

3 Preliminaries

This section gives definitions for the terms used in the work.

Definition 1. [10] A group is a nonempty set G with a binary operation "$\cdot$" that satisfies the following properties:

- Closure: $a \cdot b \in G$ for every $a, b \in G$.
- Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for every $a, b, c \in G$.
- Unity element: there exists an element $1 \in G$ such that $1 \cdot a = a \cdot 1 = a$ for every $a \in G$.
- Inverse element: for every element $a \in G$ there is an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$

Definition 2. [10] A group G is called commutative or Abelian if $a \cdot b = b \cdot a$ for every $a, b \in G$.

Definition 3. [10] A ring is a nonempty set R with two binary operations "$+$" and "$\cdot$" that satisfy the following properties:

- R is a commutative group with respect to "$+$".
- Associativity of "$\cdot$": $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for every $a, b, c \in R$.
- Distributivity: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ and $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$ for every $a, b, c \in R$.

Definition 4. [10] A commutative ring is a ring in which the operation "$\cdot$" is commutative.

Definition 5. [10] A field is a commutative ring in which the nonzero elements form a group with respect to the operation "$\cdot$".

Definition 6. [10] A finite (Galois) field is a field that contains a finite number of elements. We denote a finite field of size q as GF(q). If q is a prime, the field GF(q) coincides with the ring of integer residues modulo q.

Example 3.1 The finite field GF(2) has two elements, 0 and 1. The element 1 is the unity element of multiplication and the multiplicative inverse of itself, the element 0 is the unity element of addition. Addition and multiplication in GF(2) are described in the following tables:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Definition 7. [10] An (n, M) block code over a finite alphabet F is a nonempty subset C of size M of F^n. The parameter n is called the code length and M is the code size. The dimension (or information length) of C is defined by $k = \log_{|F|} M$.

Definition 8. [10] The elements of a code are called codewords.

Definition 9. In block coding, information words are vectors $\mathbf{v} = (v_1, v_2, ..., v_k) \in F^k$ that are encoded into codewords $\mathbf{c} = (c_1, c_2, ..., c_n) \in F^n$. The value $n - k$ is called the redundancy of the code.

Definition 10. [10] An (n, M) code C over a field $\mathbb{F} = \text{GF}(q)$ is called linear if C is a linear subspace of $\mathbb{F}^n$ over $\mathbb{F}$; namely, for every two codewords $\mathbf{c}_1, \mathbf{c}_2 \in C$ and two scalars $a_1, a_2 \in \mathbb{F}$ we have $a_1\mathbf{c}_1 + a_2\mathbf{c}_2 \in C$.
If k is the dimension of C, we say that C is a linear $[n, k]$ code over $\mathbb{F}$.

All of the codes in this work are defined over the binary field, $\mathbb{F} = \text{GF}(2)$, also denoted as $\mathbb{F}_2$.

Definition 11. [10] A generator matrix of a linear $[n, k]$ code over GF(2) is a $k \times n$ matrix whose rows form a basis of the code.

Example 3.2 Consider the following 2×3 generator matrix over GF(2):

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

The matrix G corresponds to a binary $[3, 2]$ simplex code (see Definition 18). In this code, the information word (x_1, x_2) is encoded into the codeword $(y_1, y_2, y_3) = (x_1, x_2, x_1 + x_2)$.

Example 3.3 Consider the following 3×7 generator matrix over GF(2):

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

The matrix G corresponds to a binary $[7, 3]$ simplex code. In this code, the information word (x_1, x_2, x_3) is encoded into the codeword
$(y_1, y_2, y_3, y_4, y_5, y_6, y_7) = (x_1, x_2, x_3, x_1 + x_2, x_1 + x_3, x_2 + x_3, x_1 + x_2 + x_3)$.

Definition 12. [10] Let C be a linear $[n, k]$ code over $\mathbb{F}$ and G be a generator matrix of C. We can encode information words to codewords of C by regarding the former as vectors $\mathbf{u} \in \mathbb{F}^k$ and using a mapping $\mathbb{F}^k \to C$ defined by

$$\mathbf{u} \mapsto \mathbf{u}G.$$

Since rank$(G) = k$, this mapping is one-to-one.

Definition 13. [10] A $k \times n$ generator matrix is called systematic if it has the form

$$\big(I \mid A\big),$$

where I is a $k \times k$ identity matrix and A is a $k \times (n - k)$ matrix.

Definition 14. [5] Let Σ be a finite alphabet. We say that C is a $(k, N, t, n, r)_\Sigma$ batch code over a finite alphabet Σ if it encodes any string $\mathbf{x} = (x_1, x_2, ..., x_k) \in \Sigma^k$ into n strings (buckets) of total length N over Σ, namely $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n$, such that for each t-tuple (batch) of (not necessarily distinct) indices $i_1, i_2, ..., i_t \in [k]$, the entries $x_{i_1}, x_{i_2}, ..., x_{i_t}$ can be retrieved by reading at most r symbols from each bucket. If $\Sigma = F_2$ is a finite field, we also use the notation $(k, N, t, n, r)_q$ to denote $(k, N, t, n, r)_\Sigma$.

Definition 15. [5] We say that a $(k, N, t, n, r)_q$ batch code is linear, if every entry of every bucket is a linear combination of the original database elements.

Definition 16. [2] A (k, N, t, n, r) multiset batch code is a (k, N, t, n, r) batch code which also satisfies the following property: For any multiset $i_1, ..., i_t \in [k]$ there is a partition of the buckets into t subsets $S_1, ..., S_t \subseteq [n]$ such that each symbol $x_{i_j}, j \in [t]$, can be recovered by reading at most r symbols from each bucket in S_j.

Definition 17. [2] A (k, N, t, n) (multiset) batch code is a $(k, N, t, n, 1)$ (multiset) batch code and a primitive (multiset) batch code is a (k, N, t, n) (multiset) batch code in which each bucket contains a single symbol, that is $N = n$.

In the sequel, the notation $[n, k, t]_q$ will be used to denote a linear primitive multiset batch (or PIR) code with parameters n, k and t over GF(q). The notation $[n, k, t]$ is used when $q = 2$.

Definition 18. A binary simplex code is an $[n, k]$ code where $n = 2^k - 1$. The columns of the generator matrix for such a code are made up of all nonzero binary vectors of length k.

Definition 19. [4] Let C be an $[n, k, t]_q$ batch code. It is possible to retrieve $x_{i_1}, x_{i_2}, ..., x_{i_l}$ by t different users in the primitive multiset batch code model (where the symbol x_{i_l} is retrieved by the user $l, l = 1, 2, ..., t$, respectively) if and only if there exist t non-intersecting sets $T_1, T_2, ..., T_t$ of indices of columns in the generator matrix G, and for each $T_l, 1 \leq l \leq t$, there exists a linear combination of columns of G indexed by that set, which equals to the column vector $\mathbf{e}_{i_l}^T$, for all $l \in [t]$.
Here, $\mathbf{e}_i$ denotes a vector where the element at index i is a 1 and the remaining elements are zeros.

In a batch code request, information word elements can be represented as vectors of length k, where each vector has a 1-value at the index of the corresponding information word. For example, if the information word is $\mathbf{x} = (x_1, x_2, x_3, x_4)$, then the requests x_1 and x_3 can be represented as

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

respectively.

Definition 20. A binary linear $[n, k]$ code C is called a t-functional batch code if there exists a generator matrix G for C that can serve any request consisting of t nonzero vectors in $\mathbb{F}_2^k$.

Definition 21. [9] A binary linear $[n, k]$ code C is called a t-odd batch code if there exists a generator matrix G for C that can serve any request consisting of t odd-weight vectors in $\mathbb{F}_2^k$.

Definition 22. A binary linear $[n, k]$ code C is called a t-PIR code if there exists a generator matrix G for C that can serve any request consisting of t copies of a unit vector in $\mathbb{F}_2^k$.

Definition 23. A binary linear $[n, k]$ code C is called a t-functional PIR code if there exists a generator matrix G for C that can serve any request consisting of t copies of a vector in $\mathbb{F}_2^k$.

Definition 24. [10] Let $\mathbb{F}$ be a finite field. The Hamming weight of $\mathbf{e} \in \mathbb{F}^n$ is the number of nonzero entries in $\mathbf{e}$. We denote the Hamming weight by $w(\mathbf{e})$.

Definition 25. [11] A Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets and is denoted by $\left\{ {n \atop k} \right\}$.

Definition 26. [10] Define the binary entropy function $H : [0, 1] \to [0, 1]$ by

$$H(x) = -x \log_2 x - (1 - x) \log_2(1 - x) ,$$

where $H(0) = H(1) = 0$.

Here, x is the crossover probability of the binary symmetric channel. The function describes the relation between the error probability and transmission rate of the channel, reaching its maximum value at $x = 1/2$. The binary entropy function is plotted below in Figure 1.

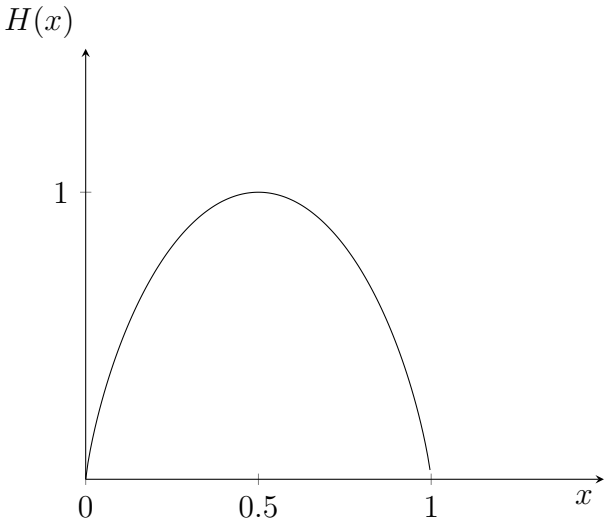

Figure 1. Binary entropy function.

Definition 27. [12] A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function $f(x)$ about a point $x = a$ is given by

$$f(x) = f(a)+f'(a)(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f^{(3)}(a)}{3!}(x-a)^3+...+\frac{f^{(n)}(a)}{n!}(x-a)^n+....$$

If $a = 0$, the expansion is known as a Maclaurin series.

Definition 28. A null matrix is a matrix whose entries are all 0.

4 Bounds on the Code Length

In this section, the properties of functional batch and PIR codes are studied. Several bounds on the length of such codes are presented .

In the general scenario of a block code, a sequence of k symbols (or bits), called an information word, is encoded into a sequence of n symbols, $n \geq k$, called a codeword. Batch codes are block codes which allow for a batch of some t information symbols to be simultaneously recovered, using the symbols of the codeword. Functional batch codes are a variant of batch codes where requests consist of t linear combinations of the information symbols. PIR codes are similar to batch codes, with the difference that a request consists of t copies of the same information symbol, or in the case of (linear) functional PIR codes, t equal linear combinations of information symbols.

The codes under study in this thesis are functional batch codes and functional PIR codes. For a given k and t, it is desirable to find the lowest possible value of n, such that any t requests can be satisfied.

An example of a functional batch code is given below.

Example 4.1 The generator matrix in Example 3.1,

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix},$$

is a functional batch code with $k = 2$, $t = 2$ and $n = 3$. An information word $x = (x_1, x_2)$ is coded into the codeword $xG = (x_1, x_2, x_1 + x_2) = (y_1, y_2, y_3)$. All possible functional batch requests for $t = 2$ (ignoring permutation of the requested elements) are as follows:

- x_1, x_1, which can be expressed as a pair of vectors $\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, respectively,
- x_1, x_2, which can be expressed as $\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}$,
- x_2, x_2, which can be expressed as $\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}$,
- $x_1 + x_2, x_1$, which can be expressed as $\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$,
- $x_1 + x_2, x_2$, which can be expressed as $\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and
- $x_1 + x_2, x_1 + x_2$, which can be expressed as $\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

The element x_1 can be recovered using y_1 or $y_2 + y_3$, because the column corresponding to y_1 is equal to the column corresponding to x_1, and so is the sum of y_2 and y_3,

$$\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}.$$

The element x_2 can be recovered using y_2 or $y_1 + y_3$. The combination $x_1 + x_2$ can be recovered using $y_1 + y_2$ or y_3. Since every request can be satisfied using two disjoint sets of codeword elements, the functional batch requirement is fulfilled. This is also the shortest possible functional batch code for $k = 2$, $t = 2$, because any code of length two cannot satisfy both requests x_1, x_1 and x_2, x_2 (both rows of the generator matrix need to contain at least two ones, but since $n = 2$, there is no way to achieve a 2×2 matrix with linearly independent rows, that meets this requirement). Since the possible functional PIR requests for $k = 2$ and $t = 2$ are pairs of identical elements or combinations, (x_1, x_1), (x_2, x_2) or $(x_1 + x_2, x_1 + x_2)$, it can be seen that G is also a functional PIR code. It is also the minimum length code for these k and t, because of the same reason as described above in the batch case.

4.1 Asymptotic Lower Bounds

This section presents asymptotic results on the optimum value of n depending on a fixed t, as k approaches infinity. Two lower bounds are presented, one for functional batch codes and another for functional PIR codes.

The following result appears as Theorem 23 in a paper by Zhang et al. [7], but the proof given here is slightly different.

Theorem 1. *Fix some integer $t > 0$. Let C be a family of $[n, k, t]$ functional batch codes, where k and n both tend to infinity. Then*

$$\lim_{k\to\infty} n(k) \geq \frac{t}{\log_2(t+1)}k. \tag{1}$$

Proof. The number of different nonzero binary vectors of length k is $2^k - 1$. Since C is a multiset functional t-batch code, the number of different requests for C is therefore $\left(\!\!\binom{2^k-1}{t}\!\!\right) = \binom{2^k+t-2}{t}$. Let G be the generator matrix of C. To satisfy a request $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t$ of t nonzero vectors in $\mathbb{F}^k$, we need t mutually disjoint sets $T_1, T_2, ..., T_t$ of indices of the columns in G. For each T_l, $1 \leq l \leq t$, there exists a linear combination of the columns indexed by that set, which is equal to $\mathbf{u}_l$. The remaining columns, which do not belong to any set T_l, form a set of unused columns. Altogether, the columns of G are divided into $t + 1$ non-empty partitions, or t non-empty partitions if there are no unused columns. The number of ways to divide the columns of G into $t+1$ non-empty partitions is equal to $\left\{ {n \atop t+1} \right\}$. For each such partition, the set of unused columns can be chosen in

$t+1$ ways. The number of ways to divide the columns into t non-empty partitions is $\left\{ {n \atop t} \right\}$. Therefore, the total number of ways to divide the columns of G, such that the result satisfies a request in C, is $(t+1)\left\{ {n \atop t+1} \right\} + \left\{ {n \atop t} \right\}$.

Since C is a functional batch code, this number is larger or equal to the number of different requests. Thus,

$$(t+1)\left\{ {n \atop t+1} \right\} + \left\{ {n \atop t} \right\} \geq \binom{2^k + t - 2}{t} .$$

By using the property $m\left\{ {n \atop m} \right\} + \left\{ {n \atop m-1} \right\} = \left\{ {n+1 \atop m} \right\}$ [13, p. 264], we obtain

$$\left\{ {n+1 \atop t+1} \right\} \geq \binom{2^k + t - 2}{t} .$$

By using $\binom{m}{v} \geq \frac{m^v}{v^v}$ [14, p. 1186] and $\left\{ {m \atop v} \right\} \geq \frac{(v^2+v+2)v^{m-v-1}}{2} - 1$ [13, Theorem 3], we obtain

$$\frac{((t+1)^2 + (t+1) + 2)(t+1)^{n+1-(t+1)-1}}{2} - 1 \geq \frac{(2^k + t - 1)^t}{t^t} .$$

By ignoring the small term of -1 in the left-hand side and by multiplying the inequality by $2t^t$, we obtain

$$t^t((t+1)^2 + (t+1) + 2)(t+1)^{n+1-(t+1)-1} \geq 2(2^k + t - 1)^t .$$

By simplifying further and by ignoring the small terms, we obtain an asymptotic inequality

$$(t+1)^t(t+1)^2(t+1)^{n-t-1} \geq 2^{kt+1} ,$$

equivalently

$$(t+1)^{n+1} \geq 2^{kt+1} .$$

By taking logarithm to the base 2, we obtain

$$(n+1)\log_2(t+1) \geq kt + 1 \text{ and thus,}$$

$$n \geq \frac{kt}{\log_2(t+1)} + \frac{1}{\log_2(t+1)} - 1 .$$

Finally,

$$\lim_{k\to\infty} n \geq \frac{kt}{\log_2(t+1)} .$$

□

A similar bound can be proved for PIR codes. The following result is analogous to Theorem 14 in the aforementioned paper by Zhang et al. [7], but the proof given here is simpler.

Theorem 2. *Fix some integer $t > 0$. Let C be a family of $[n, k, t]$ functional PIR codes, where k and n both tend to infinity. Then*

$$\lim_{k\to\infty} n(k) \geq \frac{1}{H(1/t)}k. \tag{2}$$

Proof. The number of possible different functional PIR requests of size t, which do not contain nonzero vectors, is $2^k - 1$. Let G be a generator matrix of C. Each request is satisfied by some partition of the columns of G. There are n columns in G. To satisfy a functional t-PIR request $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t$ consisting of t copies of a vector in $\mathbb{F}^k$, we need t mutually disjoint sets $T_1, T_2, ..., T_t$ of indices of the columns in G, where T_i recovers u_i, $i \in \{1, 2, ..., t\}$. The size of the smallest of these sets T_l, $1 \leq l \leq t$, is at most $\lfloor n/t \rfloor$. By counting the number of different ways to divide the columns of G into sets $T_1, T_2, ..., T_t$ by the smallest T_l, $1 \leq l \leq t$, we get $\sum_{i=1}^{\lfloor n/t \rfloor} \binom{n}{i}$ (there may be different ways of dividing the other columns besides the smallest set, such that they satisfy the same request, but we can just choose one of those equivalent options for each smallest set). Each partition of columns that satisfies a t-PIR request contains one of the sets counted in the previous sum, and C must be able to satisfy any of the possible $2^k - 1$ requests, therefore, the partitions counted by $\sum_{i=1}^{\lfloor n/t \rfloor} \binom{n}{i}$ definitely cover all t-PIR requests for C. Since C is a functional PIR code, the value of this sum is larger or equal to the number of requests, $2^k - 1$, therefore

$$\sum_{i=1}^{\lfloor n/t \rfloor} \binom{n}{i} \geq 2^k - 1 \, .$$

By simplifying the left-hand side of the inequality using
$\sum_{m\leq \alpha n} \binom{n}{m} = 2^{nH(\alpha) - \frac{1}{2}\log n + O(1)}$, where $0 < \alpha < 1/2$ [11, p. 598], we get

$$2^{nH(1/t)} \geq 2^k - 1 \, .$$

If we ignore the small term -1 on the right-hand side and take logarithm to the base 2, we get

$$\log_2 2^{nH(1/t)} \geq \log_2 2^k \, ,$$

thus

$$nH(1/t) \geq k$$

and finally

$$\lim_{k\to\infty} n \geq \frac{1}{H(1/t)}k.$$

□

4.2 Upper Bounds for Simplex-Based Constructions

This section presents constructions for functional batch and PIR codes, based on the simplex codes. The upper bounds on the code length are proved for these constructions, and these bounds are analyzed along with the lower bounds presented in Section 4.1.

The simplex-based construction in the following proof is analogous to the construction of switch codes in the work of Wang et al. [6], but the construction given here is applied for functional batch codes.

Theorem 3. *For a given batch size $t = 2^{k'-1}$, there exists a family of functional batch codes of length*

$$n = \frac{3t-1}{\log_2 t + 1} k, \tag{3}$$

for any $k \in \mathbb{N}$ divisible by k'.

Proof. Let $G_{k'}$ be a generator matrix of an $[2^{k'}-1, k']$ simplex code $C_{k'}$. The matrix $G_{k'}$ consists of every non-zero binary vector in $\mathbb{F}^{k'}$. It has been proved that $G_{k'}$ is a $2^{k'-1}$-odd batch code [9, Theorem 2.3].

Consider the following $k' \times (2^{k'} - 1 + 2^{k'-1})$ matrix

$$G_t = \left[\begin{array}{c|cccc} & 1 & 1 & \cdots & 1 \\ G_{k'} & 0 & 0 & \cdots & 0 \\ & \vdots & \vdots & \cdots & \vdots \\ & 0 & 0 & \cdots & 0 \end{array}\right].$$

The first $2^{k'} - 1$ columns of G_t make up the sub-matrix $G_{k'}$, and the remaining $2^{k'-1}$ columns are vectors $\mathbf{e}_1 = (1, 0, 0, ..., 0)$ of length k', where the first element in every vector is 1 and the rest are zeros.

Let us show that G_t is a $2^{k'-1}$-functional batch code. Every even-weight vector of length k' can be presented as the sum of an odd-weight vector of length k' and the vector $\mathbf{e}_1$. Therefore, for every t-functional batch request, we can construct a new request by replacing each even-weight vector $\mathbf{u}_e$ with the odd-weight vector $\mathbf{u}_e + \mathbf{e}_1$ and one copy of $\mathbf{e}_1$, which together add up to the original even-weight vector $\mathbf{u}_e$. Since $\mathbf{u}_e + \mathbf{e}_1$ is an odd-weight vector, it can be recovered using the sub-matrix $G_{k'}$. Evidently, the remaining vector $\mathbf{e}_1$ can be recovered by using one column from the remaining sub-matrix on the right. After finding recovery sets for each of the two vectors, we can add the two sets together to obtain a recovery set for the original even-weight vector $\mathbf{u}_e$. Since there are $2^{k'-1}$ vectors in the original functional batch request, the new request will consist of $2^{k'-1}$ odd-weight vectors (these vectors may also be in the form $\mathbf{e}_1$), which will be recovered using G'_k, and some c copies of $\mathbf{e}_1$, $c \in \{0, 1, ..., 2^{k'-1}\}$, which will be recovered using the remaining sub-matrix. Therefore, G_t can satisfy any functional batch request of size $2^{k'-1}$.

Now consider the following $dk' \times d(2^{k'} - 1 + 2^{k'-1})$, $d \in \mathbb{N}$, matrix

$$G_d = \begin{bmatrix} G_t & \mathfrak{o} & \cdots & \mathfrak{o} \\ \mathfrak{o} & G_t & \cdots & \mathfrak{o} \\ \vdots & \vdots & \ddots & \vdots \\ \mathfrak{o} & \mathfrak{o} & \cdots & G_t \end{bmatrix},$$

where $\mathfrak{o}$ denotes a $k' \times (2^{k'} - 1 + 2^{k'-1})$ null matrix. The matrix G_d is a $2^{k'-1}$-functional batch code of dimension dk' and length $d(2^{k'} - 1 + 2^{k'-1})$, $d \in \mathbb{N}$. To satisfy a request $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t$, we construct t mutually disjoint sets of columns $T_1, T_2, ..., T_t$ accordingly: for the first k' symbols of each element $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t$, we construct recovery sets $T_{1_1}, T_{2_1}, ..., T_{t_1}$, respectively, using mutually disjoint sets of columns from the first $2^{k'} - 1 + 2^{k'-1}$ columns of G_d. This is possible because the copy of G_t located in the first k' rows and $2^{k'} - 1 + 2^{k'-1}$ columns is a $2^{k'-1}$-functional batch code of dimension k'. The next k' symbols will be recovered using the next $2^{k'} - 1 + 2^{k'-1}$ columns of G_d, and so on until recovery sets have been found for the last k' symbols of each element. Then, for every $\mathbf{u}_i, i \in \{1, 2, ..., t\}$, sets $T_{i_1}, T_{i_2}, ..., T_{i_d}$ will be added together to form T_i.

Therefore, the code defined by G_d is a $2^{k'-1}$-functional batch code.

Let the length of G_t be denoted by n_t. Since $t = 2^{k'-1}$, $k_t = k'$, and $n_t = 2^{k'} - 1 + 2^{k'-1}$, we can write $k_t = \log_2 t + 1$ and $n_t = 3t - 1$. Thus, we obtain (n being the length of G_d, and k being the dimension)

$$\frac{n}{k} = \frac{dn_t}{dk_t} = \frac{n_t}{k_t} = \frac{3t-1}{\log_2 t + 1}$$

and finally,

$$n = \frac{3t-1}{\log_2 t + 1} k.$$

□

A similar result can be proved for PIR codes.

Theorem 4. *For a given request size $t = 2^{k'-1}$, there exists a family of functional PIR codes of length*

$$n = \frac{2t-1}{\log_2 t + 1} k, \tag{4}$$

for any $k \in \mathbb{N}$ divisible by k'.

Proof. Consider the same $k' \times (2^{k'} - 1)$ matrix $G_{k'}$ as in Theorem 3. The matrix $G_{k'}$ is a $2^{k'-1}$-functional PIR code. This is because for each column (vector) that is not equal to the requested vector, there exists exactly one other column in $G_{k'}$ such that their sum is equal to the requested vector. Since the pairs are one-to-one, there are exactly $2^{k'-1} - 1$

such pairs, and the remaining vector is equal to the requested vector by itself, giving $2^{k'-1}$ recovery sets for any linear combination of information word elements.

Let us construct a $dk' \times d(2^{k'}-1)$, $d \in \mathbb{N}$, matrix

$$G_d = \begin{bmatrix} G_{k'} & \mathfrak{o} & \cdots & \mathfrak{o} \\ \mathfrak{o} & G_{k'} & \cdots & \mathfrak{o} \\ \vdots & \vdots & \ddots & \vdots \\ \mathfrak{o} & \mathfrak{o} & \cdots & G_{k'} \end{bmatrix},$$

where $\mathfrak{o}$ denotes a $k' \times (2^{k'}-1)$ null matrix. Since $G_{k'}$ is a $2^{k'-1}$-functional PIR code, any PIR request of size $2^{k'-1}$ (with requested vectors of length dk') can be satisfied using G_d, with the first k' elements of each requested vector being recovered using the copy of $G_{k'}$ located in the first k' rows and $2^{k'}-1$ columns of G_d, and so on, similarly to the proof of Theorem 3.

Let the length of $G_{k'}$ be denoted by n'. Since $t = 2^{k'-1}$ and $n' = 2^{k'-1}$, we can write $k' = \log_2 t + 1$ and $n' = 2t - 1$. Thus, we obtain

$$\frac{n}{k} = \frac{dn'}{dk'} = \frac{n'}{k'} = \frac{2t-1}{\log_2 t + 1}$$

and finally,

$$n = \frac{2t-1}{\log_2 t + 1} k.$$

□

From Theorem 3, we see that the bound (3) on functional batch code length holds for the construction based on the simplex codes. Also, according to Theorem 1, the bound (1) holds for any linear batch code. Thus, we obtain that the optimum value of n satisfies

$$\frac{t}{\log_2(t+1)} k \leq n \leq \frac{3t-1}{\log_2 t + 1} k.$$

By ignoring small constants, we obtain that:

$$\frac{t}{\log_2 t} k \leq n \leq \frac{3t}{\log_2 t} k. \tag{5}$$

Thus, we can see that the simplex-based codes satisfy $n(k) = \Theta(\frac{t}{\log_2 t} k)$. Therefore, the bound in Theorem 1 is asymptotically tight.

It can also be shown that for large values of t, the bounds (1) and (2) in Theorem 1 and Theorem 2, respectively, are very close to each other. From the definition of the binary entropy function,

$$H(x) = -x \log_2 x - (1-x) \log_2 (1-x) \, .$$

For small values of x, the value of $(1-x)$ is approximately 1. The Maclaurin series (see Definition 27) of $\log_2(1-x)$ is

$$\log_2(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \ldots \, ,$$

and thus, the value of $-(1-x)\log_2(1-x)$ for small values of x is approximately x. Thus, $H(x) \approx -x \log_2 x + x$. In this expression, the dominant term is $-x \log_2 x$. In Theorem 2, $x = \frac{1}{t}$, which means that x is small when t is large. Thus, for large values of t, it holds that

$$H(1/t) \approx -\frac{1}{t} \log_2 \frac{1}{t} = -\frac{1}{t} \log_2 t^{-1} = \frac{\log_2 t}{t} \, .$$

From this, we obtain that:

$$\frac{1}{H(1/t)} k \approx \frac{1}{\frac{\log_2 t}{t}} k = \frac{t}{\log_2 t} k \, ,$$

which is very close to the result in Theorem 1, if the small value 1 in $\log_2(t+1)$ is ignored.

By using this knowledge, a similar result as (5) can be shown for the simplex-based functional PIR codes. From Theorem 4, it is seen that the bound (4) holds for the construction based on the simplex codes. According to Theorem 2 and the previously shown estimate for $\frac{1}{H(1/t)} k$, the bound

$$\lim_{k \to \infty} n \geq \frac{t}{\log_2 t} k,$$

holds for large values of t. Thus, (for large values of t) we obtain that the optimum value of n satisfies

$$\frac{t}{\log_2 t} k \leq n \leq \frac{2t-1}{\log_2 t + 1} k.$$

By ignoring small constants, we obtain that:

$$\frac{t}{\log_2 t} k \leq n \leq \frac{2t}{\log_2 t} k.$$

Therefore, the functional PIR code length of the simplex-based codes satisfies $n(k) = \Theta(\frac{t}{\log_2 t} k)$ for the functional PIR case, which shows that the bound (2) in Theorem 2 is also asymptotically tight.

5 Experimental Results

This section presents computer search algorithms for finding the smallest functional batch and PIR code lengths for a given request size and code dimension. The algorithms are applied to small values of these parameters and the results are compared with the values given by the bounds from Section 4.

5.1 Algorithms for the Computer Search of Codes

This section describes the algorithms used in searching for the smallest functional batch and functional PIR code length for given values of k and t.

Algorithm 1: Check if a functional batch (PIR) code exists for a given k, n and t

Input: $type_{integer}$ **k**, $type_{integer}$ **n**, $type_{integer}$ **t**

Result: $type_{Boolean}$ **exists**

```
1 matrices = generate matrices size k × n;
2 for matrix in matrices do
3     if matrix has at least t ones in each row then
4         if matrix satisfies all size t requests then
5             return True;
6 return False;
```

Algorithm 1 checks all possible matrices of a given size $k \times n$ until it finds one that corresponds to a t-functional batch (PIR) code, meaning that it can satisfy all t-functional batch (PIR) requests.

The following list further explains parts of this algorithm:

1. Line 1 describes generating binary matrices of size $k \times n$. This is done by generating non-zero vectors of length k and taking combinations of size n from those vectors (not accounting for permutations). To check only systematic matrices, a different function can be used which first generates the systematic portion of size $k \times k$ and then generates all remaining different sub-matrices for the remaining $n - k$ columns.

2. Line 3 describes a function that checks whether the matrix given as the argument has at least one 1 in each row. Any matrices that do not meet this requirement are skipped, because it is impossible for such matrices to satisfy a functional batch (PIR) code request where every requested vector contains a 1 at the same index.

3. Line 4 describes a function which checks whether every t-functional batch (PIR) request can be satisfied using the given matrix. The algorithm for this is shown below in Algorithm 2.

Algorithm 2: Check if a matrix satisfies every t-functional batch (PIR) request

Input: $type_{array}$ **matrix**, $type_{integer}$ **t**

Result: $type_{Boolean}$ **satisfies**

```
1 requests = generate requests length k (determined by the input matrix), size t;
2 for request in requests do
3     if matrix does not satisfy request then
4         return False;
5 return True;
```

Algorithm 2 checks whether a given matrix can satisfy all different t-functional batch (PIR) requests by generating all different requests and searching for recovery sets for each request.

The following list further explains parts of the algorithm:

1. Line 1 describes a function that returns an array of all different functional batch requests of size t by generating binary vectors of length k and taking combinations of size t (not accounting for permutations). To check whether a matrix corresponds to a functional PIR code, a different function can be used which returns arrays comprising t copies of the same binary vector, for all different binary vectors of length k.

2. Line 3 describes a function that checks whether the given request can be recovered using t disjoint sets of the columns of the matrix. The algorithm for this is described below in Algorithm 3.

Algorithm 3 is used to check if a given matrix can satisfy a given request by finding t disjoint subsets of the columns of the matrix, each recovering one of the request elements. The function starts with the first element of the request, finds all possible recovery sets for that column, then for each of those possible recovery sets, finds all possible recovery sets for the second column using the remaining columns, and so on (going through the outermost for-cycle for each request element) until all request elements have been covered.

Algorithm 3: Check if the given matrix satisfies the given request

Input: $type_{array}$ **matrix**, $type_{array}$ **request**

Result: $type_{Boolean}$ **canRecover**

```
1  partialAnswers = [[emptyArray, allMatrixIndexesArray ]];
2  for element in request do
3      initialize empty array updatedPartialAnswers;
4      for partialAnswer in partialAnswers do
5          construct submatrix from unused columns;
6          recSetsForIthCol = find recovery sets for the i-th column;
7          for recSet in recSetsForIthCol do
8              convert indexes of recSet to original matrix indexes;
9              identify remaining indexes after removing recSet indexes;
10             add the newly found partial answer to updatedPartialAnswers;
11     if updatedPartialAnswers is empty then
12         return False;
13     partialAnswers = updatedPartialAnswers;
14 if at least one set of recovery sets has been found then
15     return True;
16 return False;
```

The following list further explains parts of Algorithm 3:

1. Line 5 describes a function that removes already used columns from the matrix before searching for recovery sets for the new element, to ensure that the recovery sets for all elements will be disjoint. To do that, it uses an array of usable indexes (in the beginning this means all column indexes) which correspond to the columns of the original given matrix.

2. Line 6 describes a function that finds all different recovery sets for the i-th element of the request, using the sub-matrix that was left over after determining recovery sets for the previous request elements. This is described further below in Algorithm 4.

3. Line 8 describes a function that converts the indexes of the columns of the sub-matrix (the indexes corresponding to the newly found recovery set) to the indexes that those same columns had in the original matrix.

4. Line 9 describes a function that removes the indexes of the newly found recovery set (for the i-th column) from the list of usable column indexes, so that they will not be used for constructing recovery sets for the next element in the request.

5. Line 10 describes adding the updated partial collection of recovery sets to the list of such partial answers already found. It includes the array of recovery sets for columns up to the i-th column, along with an array of the remaining unused column indexes.

Algorithm 4: Find all different recovery sets for given vector

Input: $type_{array}$ **targetElement**, $type_{integer}$ **currentIndex**, $type_{array}$ **allColumns**, $type_{array}$ **chosenIndexes**, $type_{array}$ **foundRecSets**

Result: $type_{array}$ **allRecSets**

```
1 if currentIndex > length of allColumns then
2     return empty array;
3 initialize empty array chosenColumns;
4 add elements of allColumns at chosenIndexes to chosenColumns;
5 if sum of chosenColumns is equal to targetElement then
6     return foundRecSets + [chosenIndexes ]
7 else
8     return (recursive call with currentIndex increased and next column added to
        chosenIndexes) + (recursive call with currentIndex increased and next
        column skipped from chosenIndexes)
```

Algorithm 4 recursively searches for recovery sets among the given columns for the given request element. It returns an array containing all of the recovery sets it found for the element.

5.2 Numerical Results

This section presents the actual minimal code lengths for small values of the code parameters. The minimal length values were found by exhaustive computer search, using the algorithms described in Section 5.1. The results are compared with the theoretical bounds proved in Section 4.

5.2.1 Functional Batch Codes

Table 1. Actual smallest length of a functional batch code vs. the value given by the bounds, for various small values of t and k.

k	$t = 2$	$t = 3$	$t = 4$	$t = 5$
2	3 ; 2.52; **5**	5 ; 3.0	6 ; 3.45	8 ; 3.87
3	5 ; 3.79	6 ; 4.5	7 ; 5.17 ; **11**	10 ; 5.80
4	6 ; 5.05 ; **10**	8 ; 6.0	...	...
5	8 ; 6.31	10 ; 7.5	...	...
6	9 ; 7.57 ; **15**	...	...	...

Table 1 shows a side-by-side comparison of the actual smallest length of the functional batch code for a given t and k, alongside the value given by the lower bound described in Theorem 1 (along with the value from the upper bound for some parameter values). Each cell contains two (or three) values, separated by a semicolon. The value on the left is the minimal length for a t-functional batch code of dimension k found using the algorithms described in Section 5.1. The values in bold font are the values given by the bound in Theorem 3, for suitable code lengths. The value on the right (excluding the bold numbers) is the value of the lower bound for the corresponding t and k.

5.2.2 Functional PIR Codes

Table 2 shows a side-by-side comparison of the actual smallest length of the functional PIR code for a given t and k, alongside the value given by the lower bound described in Theorem 2 (along with the value from the upper bound for some parameter values). Each cell contains two (or three) values, separated by a semicolon. The value on the left is the minimal length for a t-functional PIR code of dimension k found using the algorithms described in Section 5.1. The values in bold font are given by the bound in Theorem 4, for suitable code lengths. The value on the right (excluding the bold numbers) is the value of the lower bound for the corresponding t and k.

Table 2. Actual smallest length of a functional PIR code vs. the value given by the bounds, for various small values of t and k.

k	$t = 2$	$t = 3$	$t = 4$	$t = 5$
2	3 ; 2.0 ; **3**	5 ; 2.18	6 ; 2.47	8 ; 2.77
3	4 ; 3.0	6 ; 3.27	7 ; 3.70 ; **7**	10 ; 4.16
4	5 ; 4.0 ; **6**	8 ; 4.36	9 ; 4.93	11 ; 5.54
5	6 ; 5.0	10 ; 5.44	...	...
6	7 ; 6.0 ; **9**	...	...	...

From the comparison tables, it can be seen that the values of the lower bounds increase fairly similarly to the actual minimal lengths, while staying below the actual value. That being said, the lower bound values are not expected to be particularly accurate for such small values of k, because the bounds are asymptotic, with k approaching infinity.

6 Conclusion

In this thesis, functional batch and functional PIR codes were studied, in particular the relation between the code dimension, the request size and the code length was investigated. Two asymptotic lower bounds on the code length for a fixed request size were presented. Two upper bounds for the constructions based on the simplex codes were proved. Analysis of the bounds showed that the lower bounds are asymptotically tight and that they are very close to the corresponding upper bounds. The theoretical bounds were compared with the actual values of the minimum code length for small parameter values, using computer search algorithms. The comparison showed that the actual lengths behaved similarly to the bounds.

Future work could include improving the search algorithms for finding actual minimum code lengths for larger parameter values, for example, by applying the batch code and PIR code searching algorithms in the thesis of Simisker [15] to the functional scenario. The multiplicative constants in the bounds can probably be improved upon, which would result in an even tighter bound. Another question is whether there are other constructions of optimal or near-optimal codes that are not related to the simplex codes.

References

[1] Y. Ishai, E. Kushilevitz, R. Ostrovsky, A. Sahai, "Batch Codes and Their Applications", Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp.262-271, Chicago, ACM Press, June 2004.

[2] A. Vardy, E. Yaakobi, "Constructions of Batch Codes With Near-Optimal Redundancy," in Proceedings IEEE International Symposium on Information Theory (ISIT), pp.1197-1201, Barcelona, Spain, July 2016.

[3] H. Asi, E. Yaakobi, "Nearly Optimal Constructions of PIR and Batch Codes," in Proceedings IEEE International Symposium on Information Theory (ISIT), pp.151–155, Aachen, Germany, June 2017.

[4] V. Skachek, "Batch and PIR Codes and Their Connections to Locally Repairable Codes", Network Coding and Subspace Designs, Editors: M. Greferath, M. Pavcevic, M.A. Vazquez-Castro, N. Silberstein, pp. 427–442, January 2018.

[5] H. Lipmaa, V. Skachek, "Linear Batch Codes," in Proceedings 4th International Castle Meeting on Coding Theory and Applications (ICMCTA), Palmela, Portugal, September 2014.

[6] Z. Wang, H. M. Kiah, Y. Cassuto, "Optimal Binary Switch Codes With Small Query Size," in Proceedings IEEE International Symposium on Information Theory (ISIT), pp.636–640, Hong Kong, June 2015.

[7] Y. Zhang, T. Etzion, E. Yaakobi, "Bounds on the Length of Functional PIR and Batch Codes", IEEE Transactions on Information Theory, vol.66, no.8, pp.4917–4934, August 2020.

[8] L. Yohananov, E. Yaakobi, "Almost Optimal Construction of Functional Batch Codes Using Extended Simplex Codes", IEEE Transactions on Information Theory, vol.68, no.10, pp.6434–6451, March 2022.

[9] H. D. L. Hollmann, K. Khathuria, A.-E. Riet, V. Skachek, "On Some Batch Code Properties of the Simplex Code", Designs, Codes and Cryptography, vol.91, pp.1595–1605, December 2023.

[10] R. M. Roth, "Introduction to Coding Theory ", Cambridge University Press, March 2006.

[11] R. Graham, D. E. Knuth, O. Patashnik, "Concrete Mathematics", 2nd edition, Addison–Wesley Professional, February 1994.

[12] E. W. Weisstein, “CRC Encyclopedia of Mathematics”, vol.3, 3rd edition, Chapman & Hall, May 2009.

[13] B. C. Rennie, A. J. Dobson, “On Stirling Numbers of the Second Kind”, Journal of Combinatorial Theory, vol.7, iss.2, pp.116-121, September 1969.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “Introduction to Algorithms”, Third Edition, The MIT Press, July 2009.

[15] M. Simisker, “Study of Optimal Linear Batch Codes”, Bachelor’s thesis at the Institute of Computer Science, University of Tartu, 2017.

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Teele Tars**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Asymptotic Bounds on the Length of Functional Batch and PIR Codes,

   supervised by Vitaly Skachek.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Teele Tars
12/08/2025