Ultra-Resilient Superimposed Codes: Near-Optimal Construction and Applications

Gianluca De Marco
University of Salerno, Italy

Dariusz R. Kowalski
Augusta University, GA, USA

Abstract

A superimposed code is a collection of binary vectors (codewords) with the property that no vector is contained in the Boolean sum of any k others, enabling unique identification of codewords within any group of k. Superimposed codes are foundational combinatorial tools with applications in areas ranging from distributed computing and data retrieval to fault-tolerant communication. However, classical superimposed codes rely on strict alignment assumptions, limiting their effectiveness in asynchronous and fault-prone environments, which are common in modern systems and applications.

We introduce Ultra-Resilient Superimposed Codes (URSCs), a new class of codes that extends the classic superimposed framework by ensuring a stronger codewords' isolation property and resilience to two types of adversarial perturbations: arbitrary cyclic shifts and partial bitwise corruption (flips). Additionally, URSCs exhibit universality, adapting seamlessly to any number k of concurrent codewords without prior knowledge. This is a combination of properties not achieved in any previous construction.

We provide the first polynomial-time construction of URSCs with near-optimal length, significantly outperforming previous constructions with less general features, all without requiring prior knowledge of the number of concurrent codewords, k. We demonstrate that our URSCs significantly advance the state of the art in multiple applications, including uncoordinated beeping networks, where our codes reduce time complexity for local broadcast by nearly two orders of magnitude, and generalized contention resolution in multi-access channel communication.

2012 ACM Subject Classification Mathematics of computing → Information theory; Theory of computation → Distributed algorithms

Keywords and phrases superimposed codes, ultra-resiliency, deterministic algorithms, uncoordinated beeping networks, contention resolution

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.65

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2506.13489

Funding Partly supported by the *Fondo di Ateneo per la Ricerca di Base* (FARB) of University of Salerno and by the National Science Center, Poland (NCN), grant 2020/39/B/ST6/03288.

Acknowledgements The authors would like to thank Ugo Vaccaro for many inspiring and valuable discussions, and are also grateful to the anonymous reviewers for the constructive comments and suggestions that helped to improve the overall quality of the paper.

1 Introduction

Superimposed codes, proposed by Kautz and Singleton in 1964 [16], are a well-known and widely used class of codes represented by a binary $t \times n$ matrix, where columns correspond to codewords. These codes have a distinctive property: for any subset of k columns from the matrix, and for any column $\mathbf{c}$ within this subset, there exists at least one row where column $\mathbf{c}$ has an entry of 1 while all other columns in the subset have entries of 0. This feature allows for the unique identification of any single column (codeword) within a subset of columns.

The significance of superimposed codes cannot be overstated. Over the past six decades, these codes have found applications in an impressively wide range of fields. For instance, they have proven instrumental in information retrieval (see e.g. [18, p. 570]), pattern matching [14, 22], learning problems [2, 17], wireless and multi-access communication [26, 15, 4], distributed coloring [19, 21].

Despite their broad applicability, classic superimposed codes have a significant limitation: they require perfectly aligned (i.e., fixed) codewords. For example, in wireless distributed communication, this alignment requires synchronization before the codes can be applied. Similarly, in distributed information retrieval, it necessitates the alignment of file descriptor codewords. However, achieving synchronization is often challenging or impractical in many distributed environments, where entities may start using codewords at arbitrary times as they wake up and join the system. Dynamic data retrieval also presents difficulties, as it may involve data arriving out of order. The lack of synchronization creates a difficulty in defining a meaningful product of two codes. Other limitations arise from a limited capacity to handle data corruption. Finally, existing codes assume that the number of superimposed codewords, k, is known. However, some applications may not provide this in advance, and standard estimation techniques – such as doubling the estimate of k – are ineffective, as consecutive code estimates can overlap due to arbitrary misalignments of codewords.

The purpose of the present work is to introduce and efficiently construct new superimposed codes that simultaneously address all the above limitations. We provide a near-optimal solution that significantly outperforms prior codes, which were designed to ensure only some or weaker properties, and demonstrate usefulness of new code's properties in various applications.

1.1 Our contributions and comparison with previous works

Before presenting the formal description in Section 2 (Definition 2), we will provide a high-level overview of our codes. We begin with the classic definition of superimposed codes, followed by an outline of the properties of our codes, including both the newly introduced and previously established properties. We will then review the relevant literature related to our codes and compare their performance to prior results. Finally, we will conclude with a discussion on how these codes can enhance performance in pertinent distributed problems, particularly in beeping networks and multiple-access channels, as well as in other contexts.

▶ **Definition 1** (Classic definition: *Problem 2* in [16]). *A (k, n)-superimposed code is formally defined as a $t \times n$ binary matrix $\mathbf{M}$ such that the following property holds:*

> *($\mathcal{I}$) For any k-tuple of columns of $\mathbf{M}$ and for any column $\mathbf{c}$ of the given k-tuple, there is a row $0 \leq i < t$ such that column $\mathbf{c}$ has 1 in row i and all remaining $k-1$ columns of the k-tuple have all 0 in row i.*

The columns of matrix $\mathbf{M}$ are called codewords *and their length t is called the* code length.

Remark. Definition 1 corresponds to the *Problem 2* formulation of superimposed codes as introduced by Kautz and Singleton [16]. Specifically, in their setting, given N codewords and a parameter m, the goal is that for any subset of at most $m+1$ codewords, no codeword is logically included in the Boolean sum of the others. Our parameters (k, n) correspond to theirs via $N = n$ and $m = k - 1$. [1]

[1] The equivalence between Problem 2 and property $\mathcal{I}$ is as follows: Property $\mathcal{I}$ requires that for any subset of k columns and any column $\mathbf{c}$ in the subset, there exists a row where $\mathbf{c}$ has a 1 and all other

1.1.1 Ultra-Resilient Superimposed Codes (URSC)

Our newly proposed ultra-resilient superimposed codes represent a substantial generalization of the classic definition, designed to provide robust performance without requiring prior knowledge of the number k of superimposed codes. We start from the following enhanced version of property $\mathcal{I}$, which ensures a better isolation of codewords; consequently, as demonstrated in Section 4 (see Algorithm 2), it allows to implement a meaningful product of codes even in the presence of arbitrary codeword misalignment and bit corruption.

> **Isolation property:** ($\mathcal{I}'$) For any k-tuple of columns of $\mathbf{M}$ and any column $\mathbf{c}$ of this k-tuple, there exists a row $0 \leq i < t$ such that column $\mathbf{c}$ has a 1 in row i, while all other $k-1$ columns of the k-tuple have 0 in rows $(i-1) \bmod t$, i, and $(i+1) \bmod t$.

At a high level, the ultra-resilient capability of our codes simultaneously ensures the following:

(a) **Shift resilience:** Even if each codeword undergoes independent and arbitrary cyclic shifts, property $\mathcal{I}'$ is still preserved. This resilience is particularly effective in uncoordinated settings where computational entities can join the system at arbitrary times (asynchronous activations).

(b) **Flip resilience:** Property $\mathcal{I}'$ remains intact even if a fraction $1-\alpha$ of unique occurrences of ones in each codeword can be flipped (the classic setting is obtained for $\alpha = 1$). This ensures robust performance under adversarial jamming in dynamic, asynchronous environments.

(c) **Universality:** The construction and application of these codes do not require prior knowledge of the parameter k, the number of concurrently active codewords. This intrinsic feature enables their applicability across dynamic and uncertain settings where k may vary or remain unknown.

Note. Our codes ensure that flip resilience is fully integrated with shift resilience, so that these properties work together with the isolation property to reinforce robustness. Specifically, flip resilience is maintained under arbitrary cyclic shifts, meaning that the isolation property $\mathcal{I}'$ holds for any shifted configuration and even if a fraction $1-\alpha$ of ones in each codeword is flipped (see Definition 2). This combined ultra-resilience provides robust protection in dynamic and adversarial environments, where both positional misalignments and bitwise corruption may occur simultaneously.

In a classic superimposed code (where in particular shift resilience is not required) the isolation property $\mathcal{I}'$ can be ensured by interleaving each row of its matrix $\mathbf{M}$ with a row of 0's. Specifically, for each row i of $\mathbf{M}$, the interleaved matrix $\mathbf{M}'$ has row $2i$ as row i of $\mathbf{M}$ and row $2i+1$ filled entirely with 0's. However, in our ultra-resilient superimposed codes, where shift resilience has to be ensured, this technique fails as the independent shifts of columns would disrupt this row-by-row alignment, resulting in overlaps that violate $\mathcal{I}'$. Thus, achieving the isolation property with shift resilience requires a much more robust construction.

In classic superimposed codes, the length t of the codewords (Definition 1) serves as a key measure of efficiency and performance. However, in the more complex scenario we address – where universality must be maintained in the presence of arbitrary misalignments

columns have 0. This ensures that $\mathbf{c}$ cannot be obtained by the Boolean sum (bitwise OR) of the other $k-1$ codewords in the subset. In particular, if such an isolating row exists, then in the sum of the other codewords, that row has a 0, while $\mathbf{c}$ has a 1, thus distinguishing $\mathbf{c}$ from the sum. Conversely, if no such row existed, $\mathbf{c}$ would be covered (bitwise) by the union of the other codewords, violating Problem 2.

of the codewords – this fixed measure becomes impractical (see Section 2 for a more detailed discussion of this challenge). To overcome this limitation, we introduce a more advanced metric, that we call ***code elongation***, that dynamically adjusts to the actual (but unknown) number of superimposed codewords in the system.

Comparative Efficiency and Performance

Our work builds on and extends foundational studies on superimposed codes, enhancing them to address the unique challenges posed by dynamic and asynchronous communication scenarios, fault tolerance, and the absence of knowledge about the number k of superimposed codewords. Our main result, stated in Theorem 8 of Section 3, is as follows:

> *We provide a Las Vegas algorithm that, in polynomial time, constructs an ultra-resilient superimposed code with a near-optimal performance (code elongation) of* $O\left(\frac{k^2}{\alpha^{2+\epsilon}}\log n\right)$.

For classic superimposed codes (i.e., those with aligned codewords and no flip resilience), the best-known existential bound is $O(k^2\log(n/k))$, established by Erdös, Frankl, and Füredi [11]. Over the years, various proofs have been presented for the corresponding lower bound, including those by D'yachkov and Rykov [10], Ruszinkó [25], and Füredi [12], all converging on an $\Omega(k^2\log_k n)$ bound. An elegant combinatorial proof of this lower bound has also been provided more recently by Alon and Asodi [2]. Porat and Rothschild [22] made a significant contribution by introducing the first polynomial-time algorithm to construct classic superimposed codes with a near-optimal length of $O(k^2\log n)$.

Recently, Rescigno and Vaccaro [23] introduced a generalized fault-tolerant version of classic superimposed codes (with flip resilience, but without shift resilience). They presented a randomized construction that achieves an average code length of $O((\frac{k}{\alpha})^2\log n)$ in polynomial time. Additionally, they demonstrated that any superimposed code supporting flip resilience requires a length of $\Omega\left(\left(\frac{k}{\alpha}\right)^2\frac{\log n}{\log(\frac{k}{\alpha})}\right)$.

An early extension addressing non-aligned codewords for use in synchronization problems was introduced by Chu, Colbourn, and Syrotiuk [5, 6], who proposed a generalized version of superimposed codes with cyclic shifts and fault tolerance. In terms of our parameters, their construction achieves an efficiency of $O((k\frac{\log n}{\log k})^3)$, assuming a fixed constant α. Another significant contribution in the pursuit of generalizing classic superimposed codes to accommodate non-aligned codewords was achieved recently by Dufoulon, Burman, and Beauquier [9] in the context of asynchronous beeping models. They introduced polynomial-time constructible superimposed codes, called Uncoordinated Superimposed Codes (USI-codes), which effectively handle arbitrary shifts of codewords (without fault tolerance), exhibiting a code length of $O(k^2n^2)$.

It is important to note that all of the above upper bounds were achieved with knowledge of k, which played a significant supportive role in the design of the respective algorithms. Notably, the lack of knowledge about k presents a challenge for construction, only when shifts are introduced. Otherwise, it suffices to construct codes tailored to fixed values of k, as is typically assumed in the literature (cf. [22]), and concatenate them for exponentially growing values of k.

With a performance of $O\left(\frac{k^2}{\alpha^{2+\epsilon}}\log n\right)$, our codes significantly outperform all prior results for non-aligned codewords, and they do so in a much more general and challenging dynamic setting. Specifically, our codes provide *a stronger isolation property* $\mathcal{I}'$ *that accommodates shift resilience, flip resilience, and universality.* This means that each codeword retains isolated 1-bits in a local neighborhood of three adjacent rows, even under arbitrary cyclic

shifts, up to a $1-\alpha$ fraction of bit flips for any $0 < \alpha \leq 1$, and without prior knowledge of the parameter k, the number of active codewords. This combination of properties is unprecedented in the field.

Additionally, our codes are the *first* to achieve *near-optimal* performance in the generalization of both shift resilience and flip resilience (fault tolerance), closely adhering to the lower bound established for *fixed, aligned* codewords [23]. Importantly, all prior constructions combining shift and flip resilience performed substantially worse, even without guaranteeing the isolation property or universality.

If we set aside fault tolerance (i.e., by considering $\alpha = 1$), our *polynomial-time constructible* codes achieve a code elongation of $O(k^2 \log n)$, which matches the best existential bound for *classic* superimposed codes by Erdös, Frank and Füredi [11] for all $k = n^{o(1)}$. Notably, our codes achieve this same bound while additionally exhibiting our isolation property with shift and flip resilience and without requiring prior knowledge of the parameter k. Moreover, it is important to note that we also almost match the fundamental lower bound $\Omega(k^2 \log_k n)$, as established in [10, 25, 12, 2], which is valid even for classic superimposed codes (Definition 1). Also, our construction yields codes with asymptotically the same length $O(k^2 \log n)$ as the best-known polynomially constructible classic superimposed codes [22], *i.e.*, codes requiring codewords to be aligned and without fault tolerance ($\alpha = 1$).

Technical novelty

Rooted in the universality concept of code elongation, our construction is efficiently achieved through a novel use of a specifically designed stochastic matrix distribution (as defined in Definition 5). We prove that this distribution satisfies a crucial property that we term the Collision Bound Property (Definition 3) with high probability (cf. Lemma 6). This property is essential for translating conditions on subsets of k columns into conditions that apply to pairs of columns, allowing us to verify and construct the ultra-resilient superimposed codes in a computationally efficient manner (cf. Lemma 4). In this way, we dramatically reduce the size of the problem space from considering all k-subsets of columns to focusing on pairs of columns and their possible shifts, see Algorithm 1. This transformation reduces the complexity of the problem to polynomial time in terms of the number of codewords n, making the construction process feasible and scalable. Once the problem space is reduced to pairs of columns, we must ensure that all desired code properties (as outlined in Definition 3) are preserved during the construction process. This involves analyzing intervals within codewords and accounting for three types of shifts and rearrangements. The proof of Lemma 6 addresses this in detail; see Section 3.3 for an overview of the challenges and methods involved, while the complete proof will appear in the full version of the paper.

1.1.2 Applications

The two main applications studied in this paper are within the contexts of Beeping networks and Contention resolution, which are outlined below.

It is important to note that, although the code is obtained by a Las Vegas randomized algorithm, the resulting codewords are fixed (i.e., non-probabilistic) and can be used reliably in deterministic algorithms. Furthermore, the randomized algorithm only needs to be run once, and the generated code can be reused as many times as needed.

In distributed applications, the code is typically pre-computed by an external authority and then provided to (or hardwired into) the nodes, similarly to the common assumption of PKI (Public Key Infrastructure) availability in secure distributed communication. This

external computation ensures that all nodes are equipped with codewords coming from exactly the same code, guaranteeing consistency across the system. Once deployed, the code can be (re-)used by nodes across any execution of any algorithm that relies on it.

Deterministic Neighborhood learning and Local broadcast in beeping networks (Section 4)

The beeping model, introduced by Cornejo and Kuhn [7], is a minimalist communication framework where nodes communicate in discrete time slots by either beeping or remaining silent, and the only feedback a node receives is whether there was a beep in the current time slot.

In the context of uncoordinated beeping networks, our novel coding tool significantly improves the time complexity for the local broadcast problem, previously addressed in [9], and related neighborhood learning. The solution in [9] achieved a time complexity of $O(\Delta^4 M)$, where Δ is the known upper bound on the maximum node degree and M is the message size.

> *Our approach (see Section 4 and Theorem 10) nearly quadratically reduces the complexity of local broadcast to a deterministic $O(\Delta^2 \log n \cdot (M + \log n))$, where n is the number of nodes.*

This improvement is made possible through our ultra-resilient superimposed codes, which enable more efficient and resilient data transmission even under adversarial jamming. More specifically, shift resilience mitigates the impact of uncoordinated activation, while isolation allows to use a product of the code with specifically designed small pieces of code that carry desired information despite of shifts of the main code.

Deterministic generalized Contention resolution on multi-access channels

In Contention resolution (CR) on a multi-access channel, k stations out of a total ensemble of n may become active, each with a packet that can be transmitted in a single time slot. The objective is to enable each of these k contending stations to successfully transmit its packet, i.e., to transmit without causing a collision with others in the same time slot. The earliest theoretical work on contention resolution dates back over 50 years, primarily with the seminal papers by Abramson [1], Roberts [24], and Metcalfe and Boggs [20]. Since then, CR has developed a long and rich history, addressing areas such as communication tasks, scheduling, fault tolerance, security, energy efficiency, game theory, and more. However, for deterministic solutions, only recently has the problem been studied in the challenging setting of arbitrary activation times, with the best known *existential* upper bound provided by De Marco, Kowalski and Stachowiak [8]. We can apply URSC codes with suitable parameters to efficiently solve an even more general CR problem, in which ***at least*** s successful transmissions per station are required; in particular, the following holds (the proof is deferred to the full version of the paper):

> *By simultaneously leveraging all three properties of our codes – shift resilience, flip resilience, and universality – we solve the generalized CR problem for any $k \leq n$ contenders (with k unknown), ensuring that each contender achieves at least s successful transmissions within $O((k + \frac{s}{\log n})^2 \log n)$ rounds after activation, for any $s \geq 1$.*

Our *constructive* upper bound matches the *existential* bound in [8] (case $s = 1$ in our generalized result) and gets very close to the lower bound of $\Omega(\frac{k^2}{\log k})$ proved in the same paper.

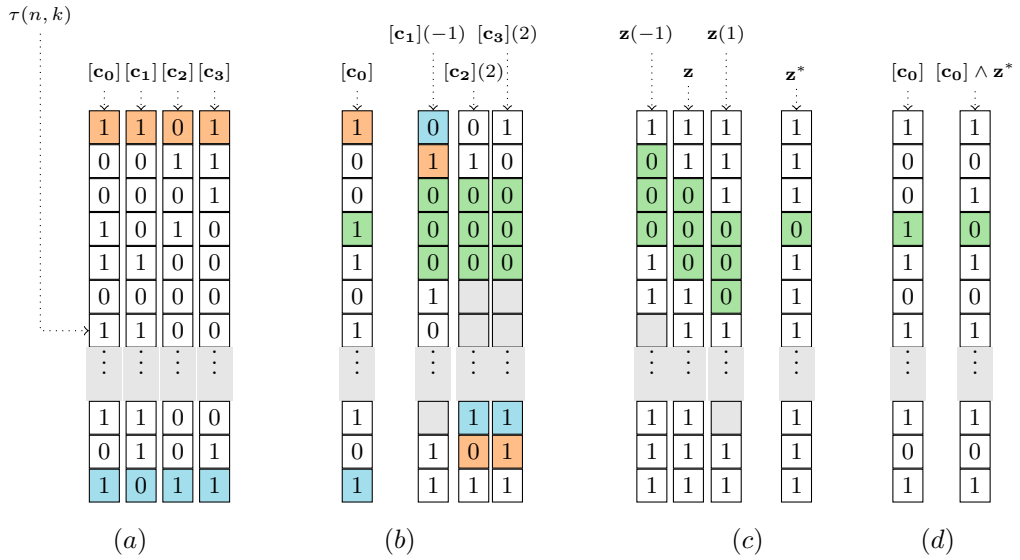

Figure 1 An illustration of the ultra-resilient properties described in Definition 2 for parameters $k = 4 \leq n$ and $\alpha = 1$. In (a) an arbitrary subset $T = \{[\mathbf{c_0}], [\mathbf{c_1}], [\mathbf{c_2}], [\mathbf{c_3}]\}$ of column vectors in the matrix is depicted. (b) shows a designated column $[\mathbf{c_j}] \in T$ (without loss of generality we assume that $[\mathbf{c_j}] = [\mathbf{c_0}]$), along with arbitrary shifts applied to the other columns in $T \setminus \{[\mathbf{c_0}]\}$. The initial and final bits of each original column vector are highlighted orange and cyan, respectively. In (c) we have the superposition vector $\mathbf{z} = [\mathbf{c_1}](-1) \vee [\mathbf{c_2}](2) \vee [\mathbf{c_3}](2)$ surrounded by $\mathbf{z}(-1)$ on its left and $\mathbf{z}(1)$ on the right. The rightmost column corresponds to the slipped vector $\mathbf{z}^* = \mathbf{z}(-1) \vee \mathbf{z} \vee \mathbf{z}(1)$. In (d) the vectors $[\mathbf{c_j}]$ and $[\mathbf{c_j}] \wedge \mathbf{z}^*$ are presented side by side. For $\alpha = 1$, the property ensures that $\left|([\mathbf{c_j}] \wedge \mathbf{z}^*)_{[0,\tau(n,k)]}\right| < \left|[\mathbf{c_j}]_{[0,\tau(n,k)]}\right|$ indicating the existence of at least one row where column vector $[\mathbf{c_j}]$ has a 1 while $[\mathbf{c_j}] \wedge \mathbf{z}^*$ has a 0, which in turn corresponds to $[\mathbf{c_j}]$ having a 1 while $\mathbf{z}(-1)$, $\mathbf{z}$ and $\mathbf{z}(1)$ all having a 0 in the same position (see the rows highlighted green in (b) and (c) and (d)).

2 Formal definition and notation

Given a binary vector $\mathbf{x} = (x_1, x_2, \ldots, x_t)$, we denote by $S(\mathbf{x})$ the set of all *cyclic shifts* of $\mathbf{x}$, that is, $S(\mathbf{x})$ contains all different binary vectors of the form $(x_{1\oplus i}, x_{2\oplus i}, \ldots, x_{t\oplus i})$, where $\oplus$ denotes addition mod t and $i = 0, \ldots, t-1$. It is clear that $1 \leq |S(\mathbf{x})| \leq t$.

For any binary vector $\mathbf{x}$, $|\mathbf{x}|$ represents the number of 1's in $\mathbf{x}$, also known as the *weight* of $\mathbf{x}$. The symbols $\vee$ and $\wedge$ denote bitwise OR and AND operators, respectively, applied to binary vectors.

Let $R = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_r\}$ be a set of binary vectors. Given R, we construct the set $S_\vee(R)$ as follows:

- First we consider all cyclic shifts for each vector in R.
- For each combination of cyclic shifts from the vectors in R, we perform a bitwise OR operation.

Formally, $S_\vee(R)$ is defined as: $S_\vee(R) = \left\{\mathbf{z} = \bigvee_{i=1}^{r} \mathbf{z}_i \;\middle|\; \mathbf{z}_i \in S(\mathbf{y}_i), 1 \leq i \leq r\right\}$.

In other words, $S_\vee(R)$ consists of all possible binary vectors obtained by taking the bitwise OR of one cyclic shift from each vector in R.

Given a $t \times n$ binary matrix $\mathbf{M}$, we refer to the c_j-th column vector of $\mathbf{M}$ as $[\mathbf{c_j}]$. For a column vector $[\mathbf{c_j}]$, we denote $[\mathbf{c_j}](i)$, where $i = 0, \ldots, t-1$, as the ith cyclic shift of $[\mathbf{c_j}]$. Formally, if $[\mathbf{c_j}] = (x_1, x_2, \ldots, x_t)$, then $[\mathbf{c_j}](i) = (x_{1\oplus i}, x_{2\oplus i}, \ldots, x_{t\oplus i})$ for $i = 0, \ldots, t-1$.

For simplicity, we extend this definition to any integer $i \geq 0$, with the understanding that i is taken modulo t throughout the paper. Finally, given any vector $\mathbf{x}$, its subvector from bit position β_1 to bit position β_2, where $0 \leq \beta_1 \leq \beta_2 \leq t$, is represented as $\mathbf{x}_{[\beta_1,\beta_2]}$.

We now present the formal definition of ultra-resilient superimposed codes. To do this, we first need to introduce two new concepts: code elongation, which generalizes the idea of code length for classic superimposed codes, and slipped vector, which is essential for preserving the isolation property.

Code elongation

In our general scenario, where codewords can undergo arbitrary adversarial shifts, the unknown parameter k introduces significant challenges that are not present in the classic case of fixed and aligned codewords. When codewords are fixed, a superimposed code for an unknown k can be constructed by concatenating superimposed codes for known, incrementally increasing parameters $k' \leq n$. However, it is well-known that when each codeword can experience an arbitrary adversarial shift, this concatenation technique, which attempts to "guess" the unknown parameter by concatenation, becomes ineffective. As a result, the concept of *code length*, which for a known k (and a given n) was a fixed value corresponding to the number t of rows in the matrix, evolves into the more general notion of *code elongation* in our much broader definition of ultra-resilient superimposed codes for unknown k. Code elongation is characterized by a function $\tau : \mathbb{N} \times \mathbb{N} \to [0, t)$ that, in addition to the given n, also depends on the unknown parameter k. This dependence enables the code to maintain its ultra-resilience properties across different (unknown) column subset sizes k.

Slipped vector

For any vector $\mathbf{z}$, we define the corresponding *slipped* vector as $\mathbf{z}^* = \mathbf{z}(-1) \vee \mathbf{z} \vee \mathbf{z}(1)$. The slipped vector is crucial for ensuring the isolation property in the definition of ultra-resilient superimposed codes given below (see Figure 1 for a graphical reference): any 1-bit in $[\mathbf{c_j}]$ that does not overlap with $\mathbf{z}^*$, i.e., in the surplus $[\mathbf{c_j}] \setminus ([\mathbf{c_j}] \wedge \mathbf{z}^*)$, indicates the existence of a row i such that $[\mathbf{c_j}]$ has a 1 at position i while $\mathbf{z}$ has 0's in positions $(i-1) \bmod t$, i, and $(i+1) \bmod t$.

▶ **Definition 2** (Ultra-resilient superimposed code)**.** *Let n be any integer. Given a function $\tau : \mathbb{N} \times \mathbb{N} \to [0, t)$, where $t \geq n$, and a real number $0 < \alpha \leq 1$, we say that a $t \times n$ binary matrix $\mathbf{M}$ is a (n, α)-ultra-resilient superimposed code (denoted (n, α)-URSC) of elongation τ, if the following condition holds:*

For any $2 \leq k \leq n$ and any subset T of column indices of $\mathbf{M}$ with $|T| = k$, and for any column index $c_j \in T$, the inequality

$$\left| \left([\mathbf{c_j}] \wedge \mathbf{z}^* \right)_{[0,\tau(n,k)]} \right| < \alpha \cdot \left| [\mathbf{c_j}]_{[0,\tau(n,k)]} \right|$$

is satisfied for all $\mathbf{z} \in S_\vee(T \setminus \{c_j\})$.

Remark. Regarding the concept of elongation, it might seem more natural to first present a result where the code length is directly expressed as a function of a fixed bound on k and afterwards combine codes for different parameter k into one (to guarantee ultra-resilience for any $k \leq n$). However, this simple approach does not work in our case, since ultra-resiliency is for *any shifts of the whole code*, not only for shifts of the code prefix corresponding to the considered value k. In other words, one needs to know the whole code in order to consider

arbitrary shifts, not only the code corresponding to specific value k. Hence our proposed approach of one universal code with elongation concept that measures its efficiency depending on variable k.

3 Construction of Ultra-Resilient Superimposed Codes (URSC)

This section focuses on the construction of ultra-resilient superimposed codes without knowing the parameter k. The objective is to design a randomized algorithm that, given the input parameters n and α and for any $\epsilon > 0$, efficiently generates an ultra-resilient superimposed code with elongation $\tau(n,k) = c(k^2/\alpha^{2+\epsilon}) \ln n$, for any (unknown) $1 < k \leq n$. This is near-optimal in view of the $\Omega((k/\alpha)^2(\log_{k/\alpha} n)$ lower bound proved in [23]. This lower bound also implies that to ensure the code elongation remains within practical limits, specifically polynomial in k, one can reasonably assume that $e^{-k} < \alpha \leq 1$.

Although the construction algorithm is randomized, the generated code can be used reliably in deterministic algorithms. (We follow the same approach of Las Vegas construction of fixed codes as in some previous literature, see e.g., [3].) Additionally, the code only needs to be generated once, as it can be reused in different contexts as long as the parameters n and α remain unchanged.

Our approach to efficiently constructing the codes revolves around two key concepts: the *Collision Bound Property* and the strategic selection of *assignment probabilities* for the 1's and 0's in the matrix. These concepts are closely intertwined: the Collision Bound Property streamlines the computational verification of code correctness, while the strategic assignment probabilities guarantee that the matrix satisfies the Collision Bound Property.

In Section 3.1, we introduce the Collision Bound Property and we demonstrate its sufficiency in ensuring that a given matrix qualifies as an ultra-resilient superimposed code. Section 3.2 introduces our random matrix construction method. Subsequently, Section 3.3 serves as the main technical segment where we establish that matrices generated using our random procedure have a high probability of satisfying both inequalities stipulated by the Collision Bound Property and consequently of qualifying as ultra-resilient superimposed codes. Finally, in Section 3.4, we outline the construction algorithm. This algorithm efficiently utilizes repeated applications of our random procedure of Section 3.2 to generate ultra-resilient superimposed codes.

Throughout this section, we assume that the two parameters n and α are fixed and given. Specifically, n is an integer such that $n \geq 2$, and α is a real number such that $e^{-k} < \alpha \leq 1$.

3.1 Collision Bound Property

The definition of URSC codes involves a condition on subsets of k columns, which can be computationally challenging to verify due to the super-polynomial number of such subsets. A crucial step towards an efficient construction is the introduction of the *Collision Bound Property*, a sufficient condition for ensuring the resilience properties of (n,α)-URSC, which applies to pairs of columns rather than subsets of k columns.

Before delving into the formal definition, let us summarize the Collision Bound Property. This property divides each column into upper and lower segments – specifically $[0, \tau_1(n,k)]$ and $[\tau_1(n,k), \tau_2(n,k)]$, with $\tau_2(n,k)$ corresponding to the elongation of the code – and ensures two key inequalities: the *Weight Inequality*, which pertains to any individual column, and the *Collision Weight Inequality*, which applies to any pair of columns.

- **Weight Inequality**: The first inequality compares the weights of the upper and lower segments of any column. Specifically, it ensures that the weight of the upper segment is at most α times the weight of the lower segment. This establishes a specific dominance of the lower segment's weight over the upper segment's weight.
- **Collision Weight Inequality**: The second inequality bounds the "collision weight" of any pair of columns:

$$\left|\Big([\mathbf{c_j}] \wedge \big([\mathbf{c_{j'}}](i-1) \vee [\mathbf{c_{j'}}](i) \vee [\mathbf{c_{j'}}](i+1)\big)\Big)_{[\tau_1(n,k),\tau_2(n,k)]}\right| ,$$

defined as the number of positions in the lower segment where a column intersects with the slipped vector of any cyclic shift of the other column of the pair. This inequality ensures that this collision weight does not exceed α times one $(k-1)$th of the lower segment's weight. This helps in controlling the overlap between columns, which is crucial for maintaining the superimposed code property.

▶ **Definition 3** (Collision Bound Property). *Let $\mathbf{M}$ be a $t \times n$ binary matrix for some integer $t \geq n$. Let $\tau_1, \tau_2 : \mathbb{N} \times \mathbb{N} \to [0, t)$ be two integer functions.*

Collision Bound Property $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$*: For every $1 < k \leq n$, for each pair of column indices c_j and $c_{j'}$ and every cyclic shift $[\mathbf{c_{j'}}](i)$, $0 \leq i \leq t-1$, both of the following inequalities hold:*

- ***Weight Inequality**:*

$$|[\mathbf{c_j}]_{[0,\tau_1(n,k)]}| \leq \alpha\, |[\mathbf{c_j}]_{[\tau_1(n,k),\tau_2(n,k)]}| , \tag{1}$$

- ***Collision Weight Inequality**:*

$$\left|\Big([\mathbf{c_j}] \wedge \big([\mathbf{c_{j'}}](i-1) \vee [\mathbf{c_{j'}}](i) \vee [\mathbf{c_{j'}}](i+1)\big)\Big)_{[\tau_1(n,k),\tau_2(n,k)]}\right| \leq \left\lfloor \frac{\alpha\, |[\mathbf{c_j}]_{[\tau_1(n,k),\tau_2(n,k)]}| - 1}{k-1} \right\rfloor . \tag{2}$$

The following lemma establishes the sufficiency of the Collision Bound Property in guaranteeing that a matrix $\mathbf{M}$ is an $(n, 2\alpha)$-URSC.

▶ **Lemma 4.** *Let $\mathbf{M}$ be a $t \times n$ binary matrix for some integers $t \geq n$. Assume that $\tau_1, \tau_2 : \mathbb{N} \times \mathbb{N} \to [0, t)$ are two integer functions such that the Collision Bound Property $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$ is satisfied. Then, the matrix $\mathbf{M}$ $(n, 2\alpha)$-URSC with elongation $\tau_2(n, k)$.*

Proof. Let us consider two functions τ_1, τ_2 and a $t \times n$ binary matrix $\mathbf{M}$ satisfying property $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$. For any $1 < k \leq n$, consider a fixed subset T of $|T| = k$ column indices, and a column index $c_j \in T$. Let $\mathbf{z} \in S_\vee(T \setminus \{c_j\})$ and $c_{m_1}, \ldots, c_{m_{k-1}}$ be the column indices in $T \setminus \{c_j\}$. By definition, $\mathbf{z} = [\mathbf{c_{m_1}}](i_1) \vee \cdots \vee [\mathbf{c_{m_{k-1}}}](i_{k-1})$, for some arbitrary shifts $0 \leq i_1, \ldots, i_{k-1} \leq t-1$. Hence, applying the distribution of conjunction over disjunction, we get

$$\begin{aligned}&([\mathbf{c_j}] \wedge \mathbf{z})_{[\tau_1(n,k),\tau_2(n,k)]} \\ &\qquad = ([\mathbf{c_j}] \wedge [\mathbf{c_{m_1}}](i_1))_{[\tau_1(n,k),\tau_2(n,k)]} \vee \cdots \vee \big([\mathbf{c_j}] \wedge [\mathbf{c_{m_{k-1}}}](i_{k-1})\big)_{[\tau_1(n,k),\tau_2(n,k)]} .\end{aligned}$$

By inequality (2) of property $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$, it follows that

$$|([\mathbf{c_j}] \wedge \mathbf{z})_{[\tau_1(n,k),\tau_2(n,k)]}| \leq (k-1) \left\lfloor \frac{\alpha\, |[\mathbf{c_j}]_{[\tau_1(n,k),\tau_2(n,k)]}| - 1}{k-1} \right\rfloor < \alpha\, |[\mathbf{c_j}]_{[\tau_1(n,k),\tau_2(n,k)]}| . \tag{3}$$

Considering also the second inequality of $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$, we have:

$$\begin{aligned} \left|([\mathbf{c_j}] \wedge \mathbf{z})_{[\tau_2(n,k)]}\right| &\leq \left|[\mathbf{c_j}]_{[0,\tau_1(n,k)]}\right| + \left|([\mathbf{c_j}] \wedge \mathbf{z})_{[\tau_1(n,k),\tau_2(n,k)]}\right| \\ &< \alpha \left|[\mathbf{c_j}]_{[\tau_1(n,k),\tau_2(n,k)]}\right| + \alpha \left|[\mathbf{c_j}]_{[\tau_1(n,k),\tau_2(n,k)]}\right| \quad \text{by (1) and (3)} \\ &< 2\alpha |[\mathbf{c_j}]_{[\tau_2(n,k)]}| \ . \end{aligned}$$

Since we have established this property for all $\mathbf{z} \in S_\vee(T \setminus c_j)$, and we have shown it holds for all possible subsets T and column indices c_j satisfying $|T| = k$, we conclude that $\mathbf{M}$ is a $(n, 2\alpha)$-URSC with elongation $\tau_2(n, k)$. ◀

3.2 Random construction

In this subsection, we present a random construction of a binary matrix. As demonstrated in the next subsection, this construction has a high probability of satisfying the Collision Bound Property $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$ and, hence, of being an $(n, 2\alpha)$-URSC in view of Lemma 4. In Theorem 8 we finally obtain an (n, α)-URSC.

In addition to n and α, the construction relies on two more parameters: an arbitrarily small constant $\epsilon > 0$ and a real constant $c > 0$. The role of ϵ is to bring the elongation of the code arbitrarily close to the asymptotic bound $O((k/\alpha)^2 \ln n)$. The constant c, if sufficiently large, ensures a high probability of successful construction, as will be demonstrated in the next section.

The goal of the random construction that we are going to describe is to assign the probability $p(r)$ of having a 1 in the rth bit of each column, for $0 \leq r < t$. The ignorance of the parameter k presents new significant challenges (in addition to those related to the arbitrary shifts), as we cannot use k in assigning the probabilities $p(r)$, nor can we use a uniform distribution. Instead, we estimate k as we descend the positions of the columns, with probabilities in the upper part tailored for smaller values of k and those in the lower part for larger values of k. This is achieved by gradually decreasing the probabilities $p(r)$ as r increases, *i.e.*, as we move down the positions of the columns. Throughout this process, we must ensure that the two inequalities of the Collision Weight Property are satisfied, creating a subtle trade-off as explained below.

The Weight Inequality requires that the decrease in the frequency of 1's is controlled such that the weight of the lower segment (from $\tau_1(n, k)$ to $\tau_2(n, k)$) dominates the weight of the upper segment (up to position $\tau_1(n, k) - 1$). Conversely, the Collision Weight Inequality requires that this dominance does not lead to an excessive number of collisions in the lower segment; specifically, the number of collisions in the lower (dominating) segment must be significantly less than the weight of each column. Additionally, this inequality must be satisfied without extending $\tau_2(n, k)$ beyond the desired elongation.

As we will show, a carefully chosen probability function that decreases according to the square root of the inverse of the column's position r, strikes the right balance among all these conflicting targets.

Finally, it is worth noting, that although each column of the matrix will have length $t = c/(\alpha^{2+\epsilon})n^2 \ln n$ (since they cannot depend on k), it will be shown later that the code still guarantees an elongation of $\tau(n, k) = c/(\alpha^{2+\epsilon})k^2 \ln n$.

▶ **Definition 5** (Random Matrix Construction $\mathcal{M}(n, \alpha, \epsilon, c)$)**.** *Let us define a random matrix $\mathcal{M}(n, \alpha, \epsilon, c)$ of n columns and $t = (c/\alpha^{2+\epsilon})n^2 \ln n$ rows, generated using the following procedure. The rth bit of each column, $0 \leq r < t$, is independently set to 1 with a probability*

given by:

$$p(r) = \sqrt{\frac{1}{\lfloor r/\ln n \rfloor + 1}} \ ,$$

and to 0 *with the complementary probability. This corresponds to each column being partitioned into* $(c/\alpha^{2+\epsilon})n^2$ *blocks of equal length* $\ln n$*, with every bit of the* b*th block, for* $0 \leq b < (c/\alpha^{2+\epsilon})n^2$*, independently set to* 1 *with a probability given by* $1/\sqrt{b+1}$*, and to* 0 *with the complementary probability.*

3.3 Satisfying the Collision Bound Property

Our next objective is to show that for $\tau_1(n,k) = \frac{c}{64}k^2 \ln n$ and $\tau_2(n,k) = \frac{c}{\alpha^{2+\epsilon}}k^2 \ln n$, any random matrix constructed as illustrated in subsection 3.2, satisfies the Collision Bound Property $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$ with high probability. (It is important to clarify that, as we will see in Section 3.4, the functions τ_1 and τ_2, which are defined in terms of the unknown k (and n), do not need to be known by the construction algorithm.) Namely, we will be proving the following.

▶ **Lemma 6.** *Fix any* $\epsilon > 0$*. Define* $\tau_1(n,k) = (c/64)k^2 \ln n$ *and* $\tau_2(n,k) = (c/\alpha^{2+\epsilon})k^2 \ln n$*, where* $c > 0$ *is a sufficiently large real constant. Let* $\mathbf{M} = \mathcal{M}(n, \alpha, c)$ *be a random matrix. For any given* $1 < k \leq n$*, a pair of column indices* c_j *and* $c_{j'}$*, and a cyclic shift* $[\mathbf{c_{j'}}](i)$ *with* $0 \leq i \leq t-1$*, the probability that the Collision Bound Property* $\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$ *does not hold is less than* $\frac{6}{c^2} \cdot n^{-8\ln\left(\frac{4}{\alpha}\right)}$*.*

The proof of Lemma 6 is quite involved, as it requires bounding the probabilities associated with satisfying the Weight Inequality and the Collision Weight Inequality separately.

While the complete technical proof of Lemma 6 is deferred to the full version of the paper, we provide here a high-level overview of the main steps involved. The Weight Inequality requires us to control the expected weights of both the upper and lower segments of each column $[\mathbf{c_j}]$. We analyze these segments as expected values over the randomized matrix construction, allowing us to bound the probability that the Weight Inequality is satisfied.

For the Collision Weight Inequality, we define a random variable to represent the number of collisions between each pair of columns $[\mathbf{c_j}]$ and $[\mathbf{c_{j'}}]$ within the lower segment: $[\tau_1(n,k), \tau_2(n,k)]$. This random variable must account for any possible cyclic shift $0 \leq i < t$ (and corresponding slipped vector) of the second column $[\mathbf{c_{j'}}]$ relative to the first column $[\mathbf{c_j}]$. Specifically, for any fixed shift $0 \leq i < t$, this random variable is given by $\sum_{d=-1}^{1} |[\mathbf{c_j}] \wedge [\mathbf{c_{j'}}](i+d)|$. Bounding the probability that this variable meets the inequality's criteria is particularly challenging for the following reasons.

The expected value of this variable is heavily influenced by the shift magnitude of the second column of the pair, which significantly affects, within the lower interval $[\tau_1(n,k), \tau_2(n,k)]$, the probabilities $p\left((r+i+d) \bmod t\right)$, representing the probability that the slipped vector of the shifted column hosts a 1 in its rth position after a shift i, for $0 \leq i < t$ and $d = -1, 0, 1$. To address this, we analyze three separate cases based on the magnitude of probabilities $p\left((r+i+d) \bmod t\right)$ for all r in the interval $\tau_1(n,k) \leq r \leq \tau_2(n,k)$.

The first case handles collisions when the shift of the second column in the pair is minimal, causing all probabilities of the shifted column to be high within the positions of the lower segment. Here, we apply the *rearrangement inequality* (see Theorem 368 in [13]), which enables a precise estimation of the expected number of collisions for high-probability entries. This technique maximizes the sum of products of corresponding probabilities (i.e., the expected collision count) by aligning the largest probability values in both columns, yielding a tight upper bound on collisions under minimal shifts.

Algorithm 1 Compute ultra-resilient superimposed codes (k is unknown).

Input: an integer $n \geq 2$, a real number $0 < \alpha \leq 1$, an arbitrarily small constant $\epsilon > 0$ and a constant $c > 0$.
Output: a matrix $\mathbf{M}$ that is a $(n, 2\alpha)$-URSC with elongation $\tau(n,k) = c(k^2/\alpha^{2+\epsilon}) \ln n$, for any $1 < k \leq n$.
Let τ_1 and τ_2 be functions defined as: $\tau_1(x,y) = (c/64)x^2 \ln y$, $\tau_2(x,y) = c(x^2/\alpha^{2+\epsilon}) \ln y$.
repeat
    Generate a random matrix $\mathbf{M} = \mathcal{M}(n, \alpha, \epsilon, c)$ (see Definition 5);
until `Check_`$\mathcal{P}$`(`$\mathbf{M}$`,` α`,` τ_1`,` τ_2`)` $=$ TRUE;
return *the generated matrix* $\mathbf{M}$;
Function `Check_`$\mathcal{P}$`(`$\mathbf{M}$`,` α`,` τ_1`,` τ_2`):`
    for $1 < k \leq n$ **do**
        for *each pair of column indices* c_j *and* $c_{j'}$ *of* $\mathbf{M}$ **do**
            for *every shift* $[\mathbf{c}_{j'}](i)$ **do**
                if *either the Weight Inequality (1) or the Collision Weight Inequality, or both, are not satisfied* **then**
                    return FALSE; `// if the Collision Bound Property is not satisfied, the function terminates and returns` FALSE
    return TRUE;

In the second case, we consider mid-range shifts where the probabilities in the shifted column remain within a constant factor of each other. To handle this, we develop a *pairwise bounding technique* that leverages the near-uniformity of probabilities across positions. This approach minimizes the dependency on exact probabilities, allowing us to obtain a balanced estimate of collisions despite moderate variations.

Finally, the third case considers the situations where the probabilities of the shifted column are very low from $\tau_1(n,k)$ up to some intermediate position τ', with $\tau_1(n,k) < \tau' < \tau_2(n,k)$, and then become large from the next position $\tau' + 1$ until $\tau_2(n,k)$. To address this, we employ a *two-segment analysis* that treats the low-probability and high-probability segments separately. Specifically, we combine the techniques from Case 2 for the low-probability segment and Case 1 for the high-probability segment.

The proof of Lemma 6 is ultimately completed by combining the probability of satisfying the Weight Inequality and that of satisfying the Collision Weight Inequality.

3.4 The construction algorithm

The construction of the ultra-resilient superimposed codes is accomplished by Algorithm 1, a randomized algorithm that, in addition to the parameter n and the real number $0 < \alpha \leq 1$, takes as input an arbitrarily small constant $\epsilon > 0$ and a constant $c > 0$. The next lemma proves that if the constant c is sufficiently large as established in Lemma 6, then Algorithm 1 outputs an $(n, 2\alpha)$-URSC with elongation $\tau(n,k) = c(k^2/\alpha^{2+\epsilon}) \ln n$, for any $1 < k \leq n$.

▶ **Lemma 7.** *For $\alpha \in (0,1]$, Algorithm 1 generates with high probability in polynomial time an $(n, 2\alpha)$-URSC with elongation $\tau(n,k) \leq \frac{c \cdot k^2}{\alpha^{2+\epsilon}} \ln n$, for any $1 < k \leq n$. The same result can be obtained in expectation.*

Proof. The algorithm sets functions $\tau_1(x,y) = (c/64)x^2 \ln y$ and $\tau_2(x,y) = c(x^2/\alpha^{2+\epsilon}) \ln y$. Then it uses a `repeat-until` loop, which at each iteration generates a random matrix $\mathbf{M} = \mathcal{M}(n,\alpha,\epsilon,c)$, according to the random construction of Definition 5, and invokes a function `Check_`$\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$. It continues iterating until this function returns TRUE.

The role of `Check_`$\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$ is simply to verify the satisfaction of the Collision Bound Property $\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$. This verification is ensured by the three nested `for` loops in lines 9 - 13. Namely, the function returns TRUE if and only if, for every $1 < k \leq n$, for every pair of column indices c_j and $c_{j'}$, and every cyclic shift $[\mathbf{c_j}](i)$ with $0 \leq i \leq t-1$, both the Weight Inequality (1) and the Collision Weight Inequality (2) hold. This indeed corresponds to the Collision Bound Property $\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$ being satisfied. Consequently, from Lemma 4 it follows that when the execution exits the `repeat-until` loop, the last generated matrix $\mathbf{M}$ is a valid (n,α)-URSC with elongation $\tau(n,k) \leq (c/\alpha^{2+\epsilon})k^2 \ln n$, for any $1 \leq k \leq n$.

To complete the proof, we must demonstrate that the total number of operations involved is polynomial with high probability and in expectation. Since the number of operations required to construct the random matrix is evidently polynomial – specifically, $n \cdot t = n \cdot \left(\frac{c}{\alpha^{2+\epsilon}} n^2 \ln n\right)$ random choices, one for each bit of the matrix – it suffices to show the following. First, we prove that each iteration requires polynomial time. Then, we establish that the number of iterations is constant both with high probability and in expectation.

Number of operations per iteration. In each iteration of the `repeat-until` loop, the function `Check_`$\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$ is invoked to verify whether the matrix satisfies property $\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$. In this function, the three nested loops cause the `if` condition on line 12 to be checked at most $n \cdot 2\binom{n}{2} \cdot t$ times. Checking the `if` condition requires verifying inequalities (1) and (2), which takes fewer than $3t$ operations.

Specifically, the first inequality involves counting the number of 1's in a vector $[\mathbf{c_j}]$ over no more than t positions and then comparing two values. The second inequality requires counting the number of 1's in a vector $([\mathbf{c_j}] \wedge ([\mathbf{c_{j'}}](i-1) \vee [\mathbf{c_{j'}}](i) \vee [\mathbf{c_{j'}}](i+1)))$ and in a vector $[\mathbf{c_j}]$, both over at most t positions, followed by comparing two values.

Overall, any iteration of the `repeat-until` loop takes time no more than $n \cdot 2\binom{n}{2} \cdot t \cdot 3t < 3n^3t^2$.

Number of iterations. The remaining task is to count the total number of iterations of the algorithm until termination. Let's consider the probability that the function `Check_`$\mathcal{P}(\mathbf{M},\alpha,\tau_1,\tau_2)$, invoked on a random matrix $\mathbf{M} = \mathcal{M}(n,c)$ with τ_1 and τ_2 set as specified in line 3, returns FALSE during a specific iteration of the innermost loop. This corresponds to the probability that, for any fixed value of k in the first loop, any pair of columns fixed in the second loop, and any shift $[\mathbf{c_{j'}}](i)$ fixed in the third loop, the inequalities (1) and (2) do not hold. Assuming that the input constant c used to build the matrix $\mathbf{M} = \mathcal{M}(n,c)$ is sufficiently large, according to Lemma 6, the probability that in any of the iterations of the three nested loops the inequalities do not hold, is at most $\frac{6}{c^2} \cdot n^{-8\ln\left(\frac{4}{\alpha}\right)}$. Therefore, by applying the union bound over all three nested loops, we can calculate the probability that the function returns FALSE in any of its iterations as follows:

$$3n^3t^2 \left(\frac{6}{c^2} \cdot n^{-8\ln\left(\frac{4}{\alpha}\right)}\right) = \frac{3c^2n^7 \ln^2 n}{\alpha^{4+2\epsilon}} \left(\frac{6}{c^2} \cdot n^{-8\ln\left(\frac{4}{\alpha}\right)}\right) = \frac{18\ln^2 n}{\alpha^6} \cdot n^{7-8\ln(4/\alpha)} < \frac{1}{n^2},$$

where the last inequality holds for $n \geq 9$. Therefore, for sufficiently large n, a single iteration of the `repeat-until` loop is sufficient to obtain a valid (n,α)-URSC with high probability. Given that we have demonstrated that each iteration requires polynomial time, it follows that the algorithm concludes within polynomial time with high probability.

Analogously, to demonstrate that the result also holds in expectation, it suffices to show that the expected number of iterations of the `repeat-until` loop is constant. The probability that the algorithm terminates at the i-th iteration is the probability that, in the first $i-1$ iterations, the function `Check_`$\mathcal{P}(\mathbf{M}, \alpha, \tau_1, \tau_2)$ returns FALSE, and only at the i-th iteration does it return TRUE. Thus, the number of iterations follows a geometric distribution with a success probability of $p = (1 - \frac{1}{n^2})$. The expected value for such a geometric distribution is known to be $1/p = \frac{n^2}{n^2-1}$, which is $O(1)$. ◀

Now we can observe that if we want to construct codes guaranteeing flip resilience for $\alpha \in (0, 1]$, it is enough to run Algorithm 1 for $\alpha' = \alpha/2$. Since $\alpha' \in (0, 1/2]$, Lemma 7 implies the following.

▶ **Theorem 8.** *For $\alpha \in (0, 1]$, Algorithm 1 can be used to generate with high probability in polynomial time an (n, α)-URSC having elongation $\tau(n, k) = O\left(\frac{k^2}{\alpha^{2+\epsilon}} \ln n\right)$, for any $1 < k \leq n$. The same result can also be obtained in expectation.*

3.5 Ultra-resilient superimposed codes parameterized by length

We could parameterize the ultra-resilient superimposed codes also by their length t, e.g., we could consider codes with length shorter than in Theorem 8. More precisely, in Definition 2, we could consider any length t of the code and vary the upper bound on the values of k for which the elongation guarantee holds, which originally was n, to some parameter Δ, where Δ is the maximum integer such that: $\Delta \leq n$ and $\tau(n, \Delta) \leq t$. This lowering of the upper bound on parameter k is natural, because shorter codes could not guarantee a successful position of any element in every configuration of the codewords, due to existing lower bounds on the length of the code even if k is known and there are no shifts, see e.g., [10].

In general, if we cut the ultra-resilient superimposed code at some smaller length $t' < t$, the resulting code may not satisfy the elongation property in Definition 2 for many values of parameter k, because the shifts in the original and the cut codes results in different configurations. However, our construction in Section 3, still works for shorter codes, with only few minor updates in the construction algorithm and in the analysis. Mainly, we need to replace the formula for the length $t = (c/\alpha^{2+\epsilon})n^2 \ln n$ by $t = (c/\alpha^{2+\epsilon})\Delta^2 \ln n$, and consider only parameters $2 \leq k \leq \Delta$. Note that throughout the analysis, all the probability bounds depend on the number of codewords n, the $\log n$ factor coming from the elongation function, and α; all of these stay the same in the t-parameterized codes, hence the probabilities stay analogous (only constants may change, which we could accomodate here by taking a larger constant c).

This way we can prove the following extension of Theorem 8 to codes with arbitrary length t.

▶ **Theorem 9.** *Algorithm 1 can be modified to generate in polynomial time, with high probability, an (n, α)-URSC of a given length t, with elongation $\tau(n, k) = O\left(\frac{k^2}{\alpha^{2+\epsilon}} \ln n\right)$, for any $1 \leq k \leq \Delta$, where Δ is the maximum integer satisfying: $\Delta \leq n$ and $\tau(n, \Delta) \leq t$. The same result can also be obtained in expectation.*

4 Code Applications in Uncoordinated Beeping Networks

4.1 Model and problem

Let G be an underlying beeping network with n nodes, modeled as an undirected simple graph. Each node v in the graph has a unique identifier from the set $\{1, \ldots, n\}$. Without

loss of generality, we will refer to both the node and its identifier as v. Each node is initially aware only of its own identifier, the total number of nodes n, and an upper bound Δ on its degree (the number of neighbors).

In the *uncoordinated setting*, nodes are activated in arbitrary rounds, as determined by a conceptual adversary.[2] The goal for each node v is to maintain a set of identifiers N_v^* that satisfies the following properties:

Inclusion: N_v^* contains all the identifiers of the neighbors of v in G.

Safety: N_v^* does not contain the identifier of any node that is not a neighbor of v in G.

This problem is referred to as *neighborhood learning*.

In a more general problem, each node has to maintain a set of input messages stored in its neighbors. This problem is often called *local broadcast*.

Time complexity of a given problem in uncoordinated beeping networks is typically measured as a worst-case (over graph topologies and adversarial wake-up schedules) number of rounds from the moment when all nodes become awaken until the task is achieved, see [9]. In the case of neighborhood learning and local broadcast, the task is achieved if all locally stored sets of neighbors (resp., messages in neighbors) contain all neighbors (resp., all neighbors' messages). Note that, due to the Safety condition, when one of these two tasks is achieved, the locally stored sets of neighbors (resp., messages) do not change in the future rounds. On the other hand, to guarantee the Inclusion property under arbitrarily long delays in adversarial wake-up schedule, algorithmic solutions have to be prepared for an arbitrarily long run – therefore, periodic algorithms seem to be most practical and as such have been considered in the literature, see [9].

It needs to be noted that the time complexity bound, denote it by $\mathcal{T}$, of our algorithms, proved in the analysis, satisfies even a stronger property: for any edge in the underlying network G, its both end nodes put each other (resp., each other's message) to the locally maintained set within time $\mathcal{T}$ after *both of them* become awaken. It means that we do not have to wait with time measurement until all nodes in the network are awaken, in case we are interested in specific point-to-point information exchange.

4.2 Neighborhood learning in uncoordinated beeping networks

Let G be an underlying beeping network of n nodes and node degree less than Δ. Each node knows only its id and the integers n and Δ. [3] All nodes have the same $(n, \frac{3}{4})$-URSC $\mathbf{M}$ from Theorem 9, of length set to $t = (c/\alpha^{2+\epsilon})\Delta^2 \ln n$ and elongation $\tau(n,k) \leq (c/\alpha^{2+\epsilon})k^2 \ln n$, for some constant $c > 0$, parameter $\alpha = \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8}$ and for any $k \leq \Delta$ (the latter is because $\tau(n,\Delta) = t$ and $\Delta \leq n$); in practice, it is enough that each node v knows only the corresponding column v of the code.[4] Additionally, each node computes in the beginning its unique block ID of length $7 + 2\log n$, defined next.

---

[2] Alternatively, the uncoordinated setting can be viewed as a temporal graph, where each node becomes "visible" after its adversarial wake-up time, and each edge becomes "visible" during the earliest time interval when both of its endpoints are awake.

[3] The algorithm and its analysis work also if each node knows some polynomial upper bound on n and a linear upper bound on Δ.

[4] Constant $\frac{3}{4}$ is set arbitrarily – it could be any constant in the interval $(\frac{1}{2}, 1)$ to allow the application of Theorem 7.

Block ID

For a given node $v \in \{1, \ldots, n\}$, we define the *block ID of* v, denoted $\mathbf{b}_v$, as follows. It is a binary sequence of length $7 + 2\log n$, it starts with three 1s followed by three 0's and another 1. To define the remaining $2\log n$ positions of the block ID, we take the binary representation of number v, of logarithmic length, and simultaneously replace each 0 by bits 01 and each 1 by bits 10.

Main idea and intuitions

Here, we provide an overview of the key ideas and intuitions behind our approach, and we refer the reader to the full version of the paper for the complete proofs. After awakening, a node periodically repeats a certain procedure – see the pseudo-code Algorithm 2. This periodicity is to assure that each node can properly pass its id, via sequence of beeps and idle rounds, to a neighbor who may be awaken at arbitrary time after the considered node. The main idea of the repeated part of Algorithm 2, see lines 6 - 16, which is in fact a form of another code, is as follows. A node v uses its corresponding codeword $\mathbf{c}_v$ from the ultra-resilient superimposed code $\mathbf{M}$ and substitutes each 1 in the codeword $\mathbf{c}_v$ by its block ID $\mathbf{b}_v$ of length $7 + 2\log n$, and each 0 in $\mathbf{c}_v$ by the block of zeros of the same length. Then, each node beeps at rounds corresponding (according to its local clock) to positions with 1's in the obtained sequence, and stays idle otherwise; see line 10 and preceding iterations in lines 7 and 8 corresponding to iterating over the length of the codewords $\mathbf{c}_v, \mathbf{b}_v$. The feedback, a beep or no beep, from the neighbors and the node itself is recorded in lines 12 and 14. Finally, a check is done (line 15) whether the recorded feedback in the last $7 + (2\log n)$ positions corresponds to any valid block ID, and if so, adding it to set N_v^* (unless it is already there), see line 16.

Intuitively, the codewords from $\mathbf{M}$ are to assure that some beeped block ID of any neighbor node will not be overlapped by any block ID beeped by its competing neighbors at the same time – therefore, it could guarantee the Inclusion property. More precisely, the shift property of the code guarantees that each neighbor has a unique 1 in its $\mathbf{c}_v$, corresponding to some block ID of v, (i.e., while other neighbors have only 0's in its overlapping blocks), regardless of the adversarial shift of the code. However, such single 1 does not guarantee passing id to the neighbor – this is why we need to substitute each 1 in the code by the block ID that allows decoding of the actual id if there are no beeping from other nodes at that time. This is challenging, because not only the original codewords from $\mathbf{M}$ could be adversarially shifted, but also the blocks themselves. Hence, to assure no beeping of other neighbors during the time when a block ID is beeped, we use the isolation property – at some point, all neighbors not only have one overlapping block of 0's, but also the preceding and the next blocks are the block of 0's. (This is because the isolation property guarantees that the preceding and the next position in the original codewords $\mathbf{c}_w$ of other at most $\Delta - 2$ neighbors have to be all 0's.)

The reason why we cannot use just a node id as its block ID is the following. The last $2\log n$ bits are to assure that any two aligned block IDs differ on at least two positions (which helps to assure that if a block ID is heard, it is not a "beeping superposition" of other block IDs but indeed a single node beeping its own block ID). The first 7 bits are to guarantee that no genuine block ID could be heard while there is no beeping block ID aligned. This assures the Safety property and helps to fulfill the Inclusion property too.

▶ **Theorem 10.** *Algorithm 2 can be instantiated with some* $(n, \frac{3}{4})$*-URSC* $\mathbf{M}$ *of length* $t = O(\Delta^2 \log n)$ *and it guarantees learning neighborhoods deterministically by each node in* $O(\Delta^2 \log^2 n)$ *rounds after awakening of the node and the neighbors.*

Algorithm 2 Neighborhood learning algorithm in an uncoordinated beeping network; pseudo-code for node v after its (uncoordinated) wake-up.

1 **Input:** Integers $n \geq \Delta \geq 1$, identifier $v \in \{1, \ldots, n\}$, a codeword $\mathbf{c}_v$ from $(n, \frac{3}{4})$-URSC $\mathbf{M}$ from Theorem 9, of length set to $t = (c/\alpha^{2+\epsilon})\Delta^2 \ln n$
2 **Maintained:** Set N_v^* of identifiers
3 $\mathbf{b}_v \leftarrow$ block ID of v ;
4 $N_v^* \leftarrow \emptyset$;
5 **while** `True` **do**
6 $\sigma_v \leftarrow$ sequence of $t \cdot (7 + 2\log n)$ zeros ;
7 **for** $i = 1$ *to* t **do**
8 **for** $j = 1$ *to* $7 + 2\log n$ **do**
9 **if** $\mathbf{c}_v[i] \wedge \mathbf{b}_v[j]$ **then**
10 v beeps `// beeping block ID of` v `when` $\mathbf{c}_v[i] = 1$
11 **if** *v has beeped or heard a beep* **then**
12 $\sigma_v[(i-1) \cdot (7 + 2\log n) + j] \leftarrow 1$
13 **else**
14 $\sigma_v[(i-1) \cdot (7 + 2\log n) + j] \leftarrow 0$
15 **if** *sub-sequence* $\sigma_v[(i-2) \cdot (7 + 2\log n) + j + 1 \mod t \cdot (7 + 2\log n), \ldots, (i-1) \cdot (7 + 2\log n) + j \mod t \cdot (7 + 2\log n)]$ *is equal to block ID of some* $w \in \{1, \ldots, n\}$ **then**
16 add w to set N_v^* (unless it is already there)

4.3 From learning neighbors to local broadcast

Now suppose that each node v has a message of length $\mathcal{M}$. It splits it into a sequence of $\log n$ messages of size $\mathcal{M}/\log n$ each, say $M_v[1], \ldots, M_v[\mathcal{M}/\log n]$. Then, it creates extended messages $M_v^*[i]$, for any $1 \leq i \leq \mathcal{M}/\log n$, as follows: it puts a binary representation of v by $\log n$ bits first, then it puts a single bit equal to 1 for $i = 1$ and to 0 otherwise (this bit indicates whether it is the first extended message or not), and then appends $M_v[i]$. The length of $M_v^*[i]$ is $2\log n + 1 \leq O(\log n)$.

There are two changes in Algorithm 2. First, we now treat $M_v^*[i]$ as a set of new ids on node v. Hence, we need a $(2n^2, 3/4)$-URSC $\mathcal{M}$ from Theorem 9, but of asymptotically same length $t = O(\Delta^2 \log n)$ as for neighborhood learning (as obviously $\log(2n^2) = \Theta(\log n)$ and we want to use the same bound Δ). Second, node v in its ith periodic procedure (recall that such a procedure is in lines 6 - 16) uses $M_v^*[i \mod \mathcal{M}/\log n]$ as its id. Let's denote this modified algorithm as the *ultra-resilient Beeping Algorithm*.

The analysis is analogous, in particular, all nodes receive all messages $M_w^*[i]$ from their neighbors w, they are able to decode that they are from w by looking at the first $\log n$ of the decoded $M_w^*[i]$, to identify the starting message by looking at bit $\log n + 1$ (and then identifying the last message from w by looking for the last bit 0 at that position before getting 1 at that position), and getting the actual content by looking at the last $\log n$ bits of $M_v^*[i]$ (and concatenating contents starting from the first identified message from w up to the last piece). The only differences are that now both the code $\mathcal{M}$ and block IDs are a constant factor longer, and node v has to wait $\mathcal{M}/\log n$ periodic procedures to be able to decode all parts of the original message. Hence, the analog of Theorem 10 can be proved for local broadcast:

▶ **Theorem 11.** *The Ultra-resilient Beeping Algorithm guarantees deterministic local broadcasting of an input message of length $\mathcal{M}$ by each node to each of its neighbors in $O(\Delta^2 \log n \cdot (\mathcal{M} + \log n))$ rounds after awakening of the node and the neighbor.*

5 Open Directions

There are several promising directions for future research. First, expanding the applications of URSCs to additional domains, such as genomic alignment and dynamic database search, could offer substantial advantages due to their inherent fault-tolerant and asynchronous properties. Exploring applications in other distributed and parallel computing contexts, as well as investigating further properties, like code weight and fairness in mechanism design, are intriguing directions for advancing URSC capabilities.

Closing the small gap between the code length of URSCs and the theoretical lower bound is a challenging yet valuable endeavor. This, along with deeper exploration of ultra-resilient properties in new settings, holds potential for further enhancing the impact of URSCs.

In the beeping network application studied in this paper, following [9], we assumed that the algorithm a priori knows an upper bound on the maximum node degree Δ, and the complexity of the communication problem depends on that upper bound. It remains an open problem whether there are solutions that are efficient when no bound on Δ is a priori known.

In summary, our contributions position URSCs as a robust, scalable solution to foundational challenges in distributed computing. We anticipate that URSCs will stimulate future research into resilient coding for asynchronous and dynamic environments, pushing the boundaries of fault-tolerant coding theory.

References

1 Norman Abramson. The aloha system: Another alternative for computer communications. In *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, AFIPS '70 (Fall), pages 281–285. ACM, 1970. doi:10.1145/1478462.1478502.

2 Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics*, 18(4):697–712, 2005. doi:10.1137/S0895480103431071.

3 Mahdi Cheraghchi and Vasileios Nakos. Combinatorial group testing and sparse recovery schemes with near-optimal decoding time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1203–1213, 2020. doi:10.1109/FOCS46700.2020.00115.

4 Marek Chrobak, Leszek Gasieniec, and Wojciech Rytter. Fast broadcasting and gossiping in radio networks. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 575–581. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892325.

5 W. Chu, C.J. Colbourn, and V.R. Syrotiuk. Slot synchronized topology-transparent scheduling for sensor networks. *Computer Communications*, 29(4):421–428, 2006. Current areas of interest in wireless sensor networks designs. doi:10.1016/J.COMCOM.2004.12.026.

6 Wensong Chu, Charles J. Colbourn, and Violet R. Syrotiuk. The effects of synchronization on topology-transparent scheduling. *Wireless Networks*, 12(6):681–690, 2006. doi:10.1007/S11276-006-6528-Z.

7 Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2010. doi:10.1007/978-3-642-15763-9_15.

8 Gianluca De Marco, Dariusz R. Kowalski, and Grzegorz Stachowiak. Deterministic non-adaptive contention resolution on a shared channel. *Journal of Computer and System Sciences*, 133:1–22, 2023. doi:10.1016/j.jcss.2022.11.001.

9 Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Can uncoordinated beeps tell stories? In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, pages 408–417. Association for Computing Machinery, 2020. doi:10.1145/3382734.3405699.

10 A.G. D'yachkov and V.V. Rykov. Bounds on the length of disjunct codes. *Problems of Information Transmission*, 18(3):7–13, 1982.

11 P Erdös, P Frankl, and Z Füredi. Families of finite sets in which no set is covered by the union of two others. *Journal of Combinatorial Theory, Series A*, 33(2):158–166, 1982. doi:10.1016/0097-3165(82)90004-8.

12 Z. Füredi. On r-cover-free families. *Journal of Combinatorial Theory, Series A*, 73(1):172–173, 1996. doi:10.1006/JCTA.1996.0012.

13 G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 1934.

14 Piotr Indyk. Deterministic superimposed coding with applications to pattern matching. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 127–136. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646101.

15 Piotr Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 697–704. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545475.

16 W. H. Kautz and R. C. Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10:363–377, 1964. doi:10.1109/TIT.1964.1053689.

17 Thomas Kesselheim, Robert D. Kleinberg, and Rad Niazadeh. Secretary problems with non-uniform arrival order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 879–888. ACM, 2015. doi:10.1145/2746539.2746602.

18 Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., USA, 1998.

19 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

20 Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM*, 19(7):395–404, July 1976. doi:10.1145/360248.360253.

21 R.A. Moser and G. Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

22 Ely Porat and Amir Rothschild. Explicit nonadaptive combinatorial group testing schemes. *IEEE Trans. Inf. Theory*, 57(12):7982–7989, 2011. doi:10.1109/TIT.2011.2163296.

23 Adele A. Rescigno and Ugo Vaccaro. Improved algorithms and bounds for list union-free families. *IEEE Trans. Inf. Theory*, 70(4):2456–2463, 2024. doi:10.1109/TIT.2023.3316435.

24 Lawrence G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, April 1975. doi:10.1145/1024916.1024920.

25 M. Ruszinkó. On the upper bound of the size of the r-cover-free families. *Journal of Combinatorial Theory, Series A*, 66(2):302–310, 1994.

26 J. Wolf. Born again group testing: multiaccess communications. *IEEE Transactions on Information Theory*, 31(2):185–191, 1985. doi:10.1109/TIT.1985.1057026.