Ben-Gurion University of the Negev

Faculty of Engineering Science

School of Electrical and Computer Engineering

Dept. of Electrical and Computer Engineering

Fourth Year Engineering Project

Final Report

Research project - Coding schemes for constrained systems

Project number:	**p-2024-002**
Students :	**Yehonatan Arama & Yuval Yacov Said**
(name & ID):	**207938903 206921892**
Supervisors:	**Ohad Elishco**
Sponsors:	
Submitting date:	

Table Of Contents

Research project - Coding schemes for constrained systems:

Student Names: Arama Yohonatan, Said Yaakov Yuval.
Email: yoni1arama@gmail.com, yuvalisaid@gmail.com
Adviser's name: Ohad Elishko.

1)Project completion summary

Background:

The project addresses the problem of encoding in systems under constraints of preventing local repetitions.

Purpose:

To enhance the efficiency of encoding schemes for generating "Repeat free codes" in various applications.

Objectives:

To develop a decodable encoding algorithm for the "K,M repeat free" problem and implement it optimally.

Innovation:

Creation of an encoding algorithm for the "K,M repeat free" problem (not previously done).

Proposed Method:

The method involves a two-phase encoding process where the initial sequence is compressed to meet constraints and then expanded back to the original length while ensuring it remains free of repetitions.

Expected Results:

The expectation is that the algorithms will successfully encode “Repeat free codes” with high efficiency and minimal error rates.

Keywords:

Repeat free codes, encoding algorithms, K,M repeat free, data compression, constrained coding, elimination algorithm.

Research project - Coding schemes for constrained systems:

שמות הסטודנטים: יהונתן ארמא, יובל יעקב סעיד.

אימייל: yoni1arama@gmail.com, yuvalisaid@gmail.com

שם המנחה: אוהד אלישקו.

2) Project completion summary (Hebrew)

1.1) רקע כללי:

הפרויקט מתעסק בבעיה של קידוד במערכת תחת אילוצים של מניעת חזרות מקומיות.

1.2) מטרה:

לשפר את היעילות של סכמות קידוד ליצירת קודים ללא חזרות (Repeat free codes) ביישומים שונים.

1.3) יעדים:

מציאת אלגוריתם קידוד בר-פענוח לבעיית ה-(K,M repeat free) ומימושו בצורה מיטבית.

1.4) החידוש:

יצירת אלגוריתם קידוד לבעיית ה-(K,M repeat free) (לא נעשה בעבר).

1.5) השיטה המוצעת:

השיטה כוללת תהליך קידוד דו-שלבי שבו הרצף הראשוני נדחס כדי לעמוד בהגבלות, ואז מורחב בחזרה לאורך המקורי תוך שמירה על היותו חופשי מחזרות.

1.6) תוצאות מצופות:

הציפייה היא שאלגוריתמים יצליחו בקידוד של קודים ללא חזרות (Repeat free codes) עם יעילות גבוהה ושיעורי שגיאות מינימליים.

1.7) מילות מפתח:

Repeat free codes, אלגוריתמי קידוד, K, M repeat free , דחיסת נתונים, קידוד מאולץ, אלגוריתם אלימנציה.

3) Project goals

Project Goal 1:

Feature: Develop encoding algorithm that can generate “K, M repeat free” sequences with a specified minimal redundancy.

Measure of Success: The creation of an efficient algorithm that can handle m’s with scale of O (log (n)) and k’s with scale of O (log (log (n))), with reduced redundancy.

Project Goal 2:

Feature: Implement the encoding algorithm efficiently through a computer program.

Measure of Success: The success of the implementation will be measured by the algorithm's performance: its ability to handle large values of n (thousands) and execute within a reasonable runtime.

4) Introduction

The high energy consumption required to maintain silicon-based data severs today has led to research into alternative approaches for data storage. One of the approaches currently being researched is DNA-based data storage. This method, however, presents several challenges and constraints that need to be addressed for effective data storage and retrieval.

One critical constraint is ensuring the storage of information without repetitions, which is essential for accurate data decoding when reading from DNA. In this project, we extend a previously proposed algorithm to address these constraints in coding.

Previous works in the area, as detailed in the paper [1], primarily focus on constraints that prevent repetitions throughout the entire length of the sequence. In this study, we relax these constraints and propose an efficient solution that ensures no repetitions within specified windows.

The problem definition in this approach is as follows: Given a sequence of length n and a pair of numbers m and k, the sequence is encoded into a different sequence such that within every window of size m, there are no repeated sub-sequences of length k such that it can be recovered to the original sequence.

5) Final Technical Specifications

Our project has four key deliverables:

1. Computer Program for Sequence Generation and Validation

This program generates random sequences and checks their compliance with the language constraints, aiming to provide intuition about the language's capacity.

2. Academic Document

This document contains formal notations and definitions, the proposed algorithm, and proof of its correctness.

3. Implementation of the Encoding Algorithm

A computer program that implements the algorithm described in the document.

4. Decoding Algorithm

A program that decodes sequences to provide additional proof of the algorithm’s correctness.

Detailed Explanation:

1. Computer Program

The computer program is written in C language and runs on a Linux OS. It offers a menu with the following options:

a. Check Manual Entered Word:

This option allows the user to input a word manually, along with values for m and k , and the program returns whether the entered word meets the constraints.

b. Choose Parameters for Random Sequence:

This option allows the user to input parameters for the language (m, n, k) and the number of iterations. The program then generates sequences under the defined conditions and checks if the words belong to the language. The program outputs the percentage of generated words that meet the constraints.

c. Automated Simulation:

This option allows the user to input two parameters and "run" over the third parameter. The output is a graph where the X-axis represents the parameter varied and the Y-axis represents the percentage of words that meet the language constraints.

2. Academic Document

The academic document, attached below.

The purposed algorithm (taken from the paper):

Algorithm 1 Encoder for $L_{km}(n)$ language

Input: Sequence $w \in \Sigma^{n-(\log\log\log(n)+1)}$

Output: Sequence $\overline{w} \in L_{km}(n)$ with $m = \log(n), k = \log\log(n) + 2\log\log\log(n) + 10, k' = \log\log(n) + \log\log\log(n) + 7$

First procedure:

$\overline{w}$ = *removingRepeatedZeros*$(\log\log\log(n), w)$

Second procedure:

while (i, j) is a *primal k'm identical window* in $\overline{w}$ **do**

  if $(\overline{w}_{j+k'+[\log\log\log(n)]} = 0^{\log\log\log(n)})$ **then**

    $\overline{w} = \overline{w}_{[j]} \circ g(j-i) \circ \overline{w}_{j+k'+[\log\log\log(n)+1]} \circ 1 \circ \overline{w}_{[-|\overline{w}|+j+k'+\log\log\log(n)+1]}$

  else

    $\overline{w} = \overline{w}_{[j]} \circ g(j-i) \circ \overline{w}_{[-|\overline{w}|+j+k']}$

  end if

end while

Third procedure:

Set $\overline{w} = \overline{w} \circ 1 \circ 0^{\log\log\log(n)+1}$

while $|\overline{w}| < n$ **do**

  Set $B = \mathrm{Supp}\left(\mathrm{fr}^{k}_{\overline{w}_{[-m]}}\right) \bigcup \bigcup_{1 \leqslant i \leqslant k-1} \mathrm{Cr}_k(\overline{w}_{[-i]})$.

  Set $S = \Sigma^k \backslash B$ and find $u \in S$

  Set $\overline{w} = \overline{w} \circ u$

end while

Return $\overline{w}_{[n]}$

3. Encoding Algorithm Implementation

The program is written in C language. Upon execution, it generates a sequence of defined length (n=65,536, m=256, k=18) encodes it according to the proposed algorithm, and produces three files: the original sequence, the encoded sequence, and a log file.

4. Decoding Algorithm Implementation

The program is written in C language. Upon execution, it searches for a specific file containing an encoded sequence, decodes it, reconstructs the original sequence, and generates a file with the original sequence.

* For programs 3 and 4, we have created additional identical programs with different parameters, but the described ones are the most challenging in terms of coding and algorithm application.

6) The approach taken to the solution and the engineering design

Planning the Deliverables:

First, as previously detailed, it was necessary to gain intuition about possible values for the solution, specifically the relationships between the values of (n), (k), and (m) that can be used to develop the algorithm. This was accomplished through the first program. After examining solutions to similar problems, it was decided that the relationships on which the algorithm would be based are: (m = O(log(n))) and (k=O(log(log(n)))). Following this, the algorithm was developed and the implementing programs were created.

Detailed Planning of Deliverables:

1. Computer Program for Sequence Generation and Validation:

- Initially, it was understood that the program would operate on very long sequences and perform repeated iterations to find repeating subsequences. Therefore, it was decided to write the program in C to ensure the shortest possible runtime, despite the challenge of writing a complex program in this language.

- The algorithm used in the program to reduce runtime is as follows:

  - Create a map for all (2^k) possible values of (k).

  - Iterate over the entire length of the sequence.

  - In each iteration, for each bit, perform an (O(1)) operation to calculate the new value of (k) relative to the previous (k).

  - In the position of this value in the map, store the current position in the sequence.

  - Calculate the last value at this position in the map relative to the current value. If the value is less than (m), a repetition is found and the value is returned.

- In this case, the runtime for the entire sequence is (O(n)).

- In any other case, the runtime for the sequence is O(m), with the naive runtime being O(m*n*k).

2. Academic Document:

- The academic document relies heavily on previous papers in the field. It details an algorithm divided into three parts:

- Part One - Marker Removal: This part is based on a proven theorem stating that a specific sequence can be removed from a word as long as the word is long enough. This operation is performed to remove all sequences of $0^{loglogloglog(n)}$. From this point, this sequence is used as a marker.

- Part Two - Elimination: In this part, there is a loop that continues until the word meets the constraint. In each iteration, the algorithm finds two identical subwords (repeats) and replaces one with a shorter sequence that can be reconstructed. Further technical details are provided in the attached document.

- Part Three - Expansion: In this part, the algorithm adds subwords to the word until it reaches the desired length.

- Note that in this part, there is an unresolved error. However, even without this part, the algorithm still provides coding that meets the constraints (the only drawback is that the length of the returned word may be shorter than the received word).

- After the algorithm, a mathematical proof of the algorithm's correctness is provided:

- The algorithm returns a word meeting the constraints.

- The algorithm terminates.

3. Implementation of the Encoding Algorithm in a Computer Program:

- Here too, it was understood that runtime is a critical factor, and therefore the program was written in C to minimize runtime.

- This program generates a sequence of 65,536 bits and writes them to a file.

- It then proceeds to the encoding stage:

- In the encoding stage, similar to the algorithm in the document, a loop continues as long as there is a repetition under the appropriate conditions.

- The repetition check is performed similarly to the algorithm in the first program, which checks whether the word belongs to the language (the algorithm with the map of size 2^k . Recall that this algorithm reduces our runtime to linear time.

- In each iteration, a repetition is replaced with a subword as described in the document.

- Note: In the document, the replaced word must also be free of repetitions, and therefore it undergoes manipulation. The implementation of this manipulation is left to the implementer's discretion and is not detailed in the document.

- This algorithm implementation includes the constrained coding of an 8-bit word to a 9-bit word without a sequence of three consecutive zeros. The encoding algorithm is stated in appendix B.
- Additionally, logs are collected in each iteration to document the actions performed during encoding.
- At the end of the encoding process, a file with the new (encoded) word and a log file are generated. This is the output to the user.

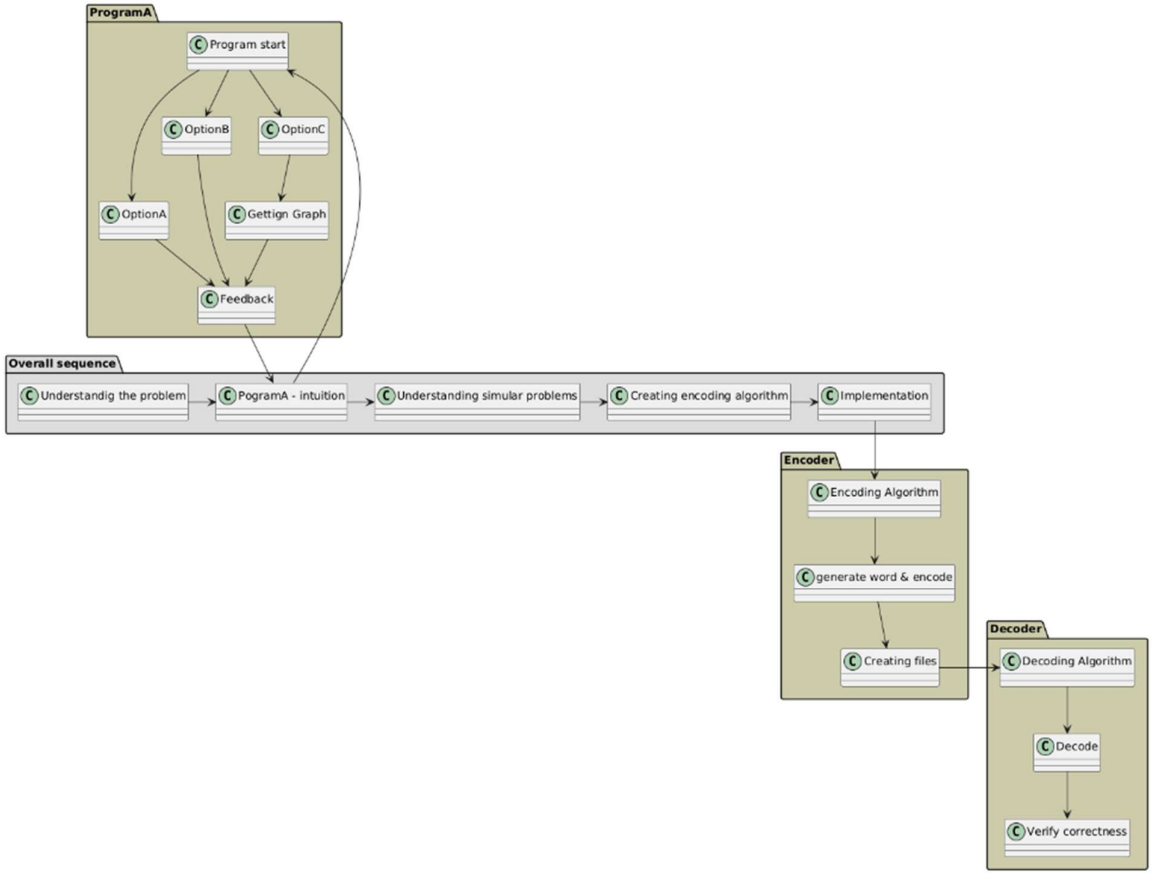
4. Implementation of the Decoding Algorithm in a Computer Program:

Again, the C language was used for the same reasons. This program searches for a specific file in the directory and performs decoding on it. The decoding process is carried out as follows:
-The algorithm scans the word and searches for a marker. Once the marker is found:
- The algorithm decodes the subword at the marker's location (recall that the marker was added when there was a repetition during the encoding stage).
It should be noted that at this stage, the constrained encoding algorithm presented earlier is also decoded.

At the end of the process, the program generates a file with the decoded word.

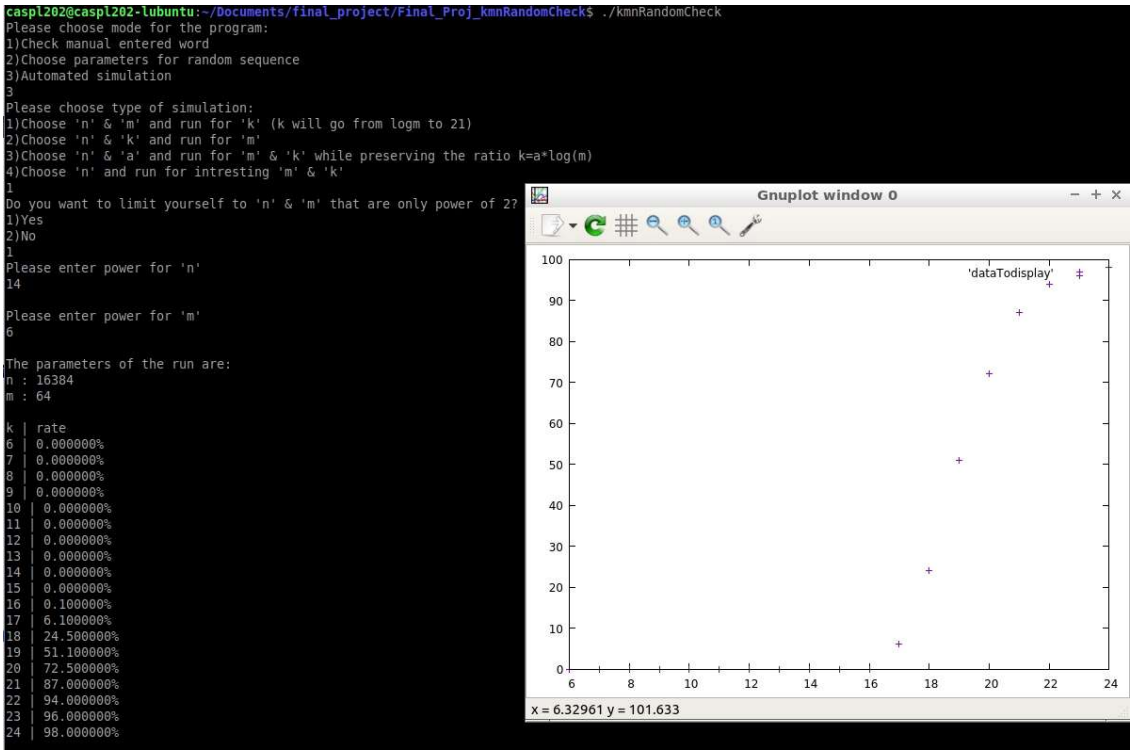
The runtime of this program is very short (a few seconds). The runtime is linear, neglecting basic C language functions.

System diagram

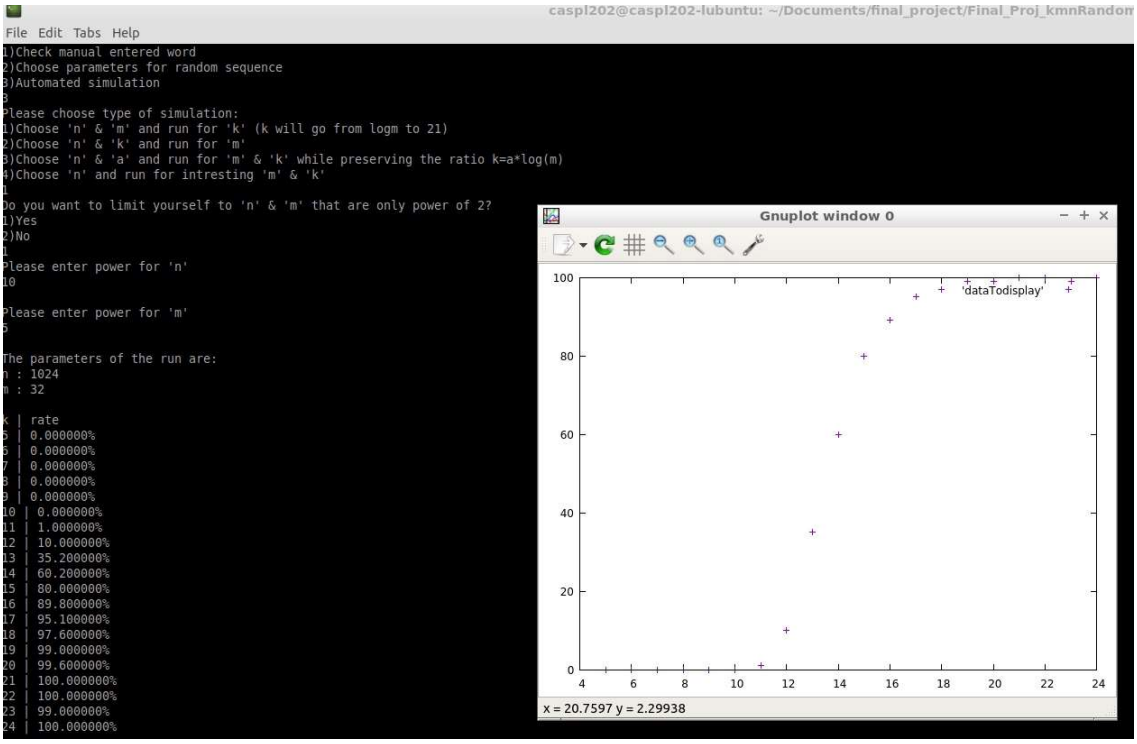


7) Final Acceptance Tests for the Product

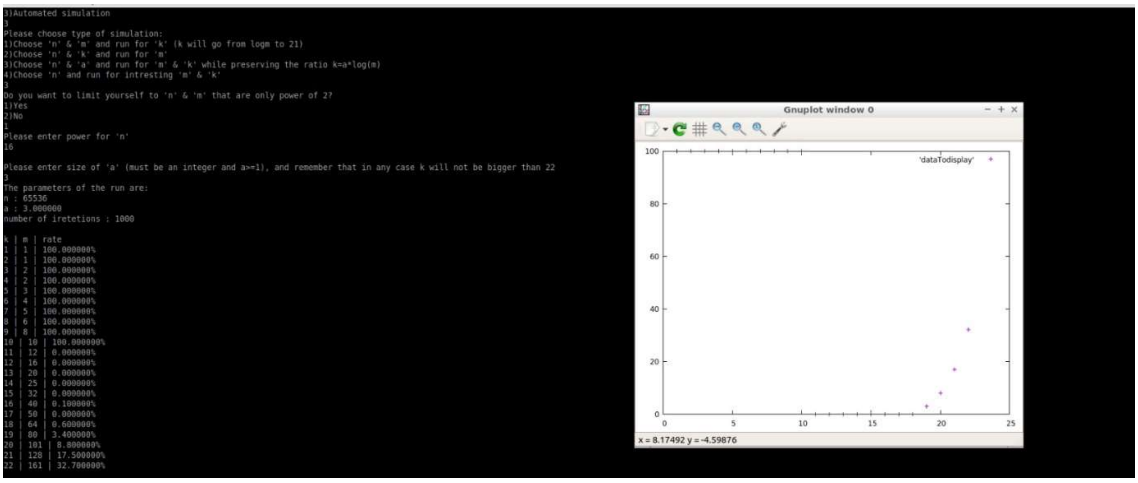
1) For the First Program: Graphs were generated to provide intuition about the "volume" of the language. For example, in the following figure, a simulation was performed for (n=2^(14)=16,384) and (m=2^6=64), resulting in the following graph:



Another example of a simulation was performed for (n=2^(10) =1024) and (m=2^5=32), resulting in the following graph:



As expected, the graphs show an increase with the rise of (k). And the language begins to “fill” in orders of scale (k=log(m)) plus some constant overhead. Consider another simulation the program can perform: A simulation for (n=2^(16) =65,536) and a constant (a=3) defining the relationship between (k) and (m) as follows: (k = a* log(m)).



Here too, it can be observed that the language "fills up" at higher values. For program performance, the following simulation was run:

For (n=2^(12) =4096), (m=2^6=64), and (k=15), 1000 iterations were performed. The results and timings are shown in the following graph:

```
caspl202@caspl202-lubuntu:~/Documents/final_project/Final_Proj_kmnRandomCheck$ time ./kmnRandomCheck
Please choose mode for the program:
1)Check manual entered word
2)Choose parameters for random sequence
3)Automated simulation
2
Do you want to limit yourself to 'n' & 'm' that are only power of 2?
1)Yes
2)No
1
Please enter power for 'n'
12

Please enter power for 'm'
6

Select which number do you want to insert:
1)'k'
2)'a'
1
Please enter size of 'k'
15

Please enter number of iterations (must be an integer and bigger than  0)
1000
The parameters of the run are:
n : 4096
m : 64
k : 15
number of iretetions : 1000

The success rate is:
4.500000%

real    0m7.039s
user    0m0.211s
sys     0m0.007s
```

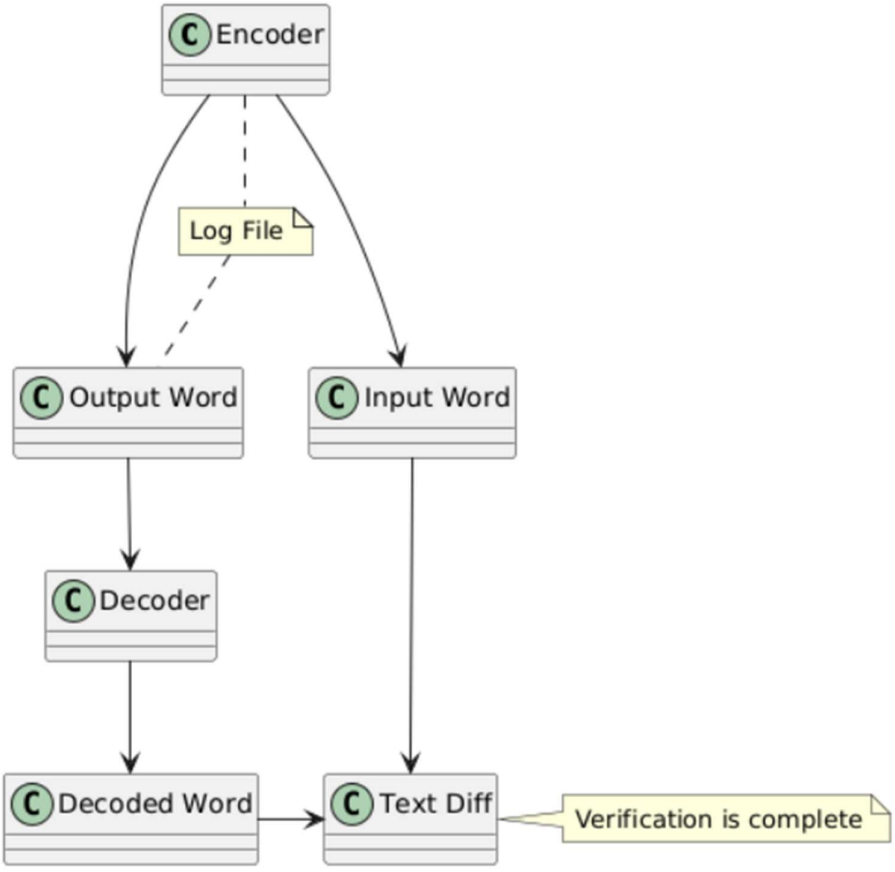
It can be seen that the runtime is very short (about 7 seconds), including the time for inputting parameters into the program. The total runtime is significantly shorter than this.

2) For the Academic Document and Proposed Algorithm:

As attached in appendix A, the algorithm is proposed with proof of correctness. In addition to the proof of correctness, further validation was performed by implementing the algorithm and its decoding.

3) For the Encoding and Decoding Algorithms:

The correctness of the algorithm and programs was validated using the following process:



Note:

There is a minor discrepancy between the output of the decoding program and the input of the encoding program. The difference is a few bits (an error of one bit per several thousand). We hypothesize that the error arises from an incorrect implementation in the case where the repetition is created to the left of the new word (a specific and exceptional case) and we need to decode the next repetition before the current one. Our hypothesis is based on the fact that the number of bits with errors matches the probability of such a case occurring, and because the encoding and decoding programs frequently handle the copying of subwords to specific locations in the word, it is possible that an error occurred in copying to the wrong location.
For the performance of the programs:

Encoder:

Decoder:

8) Challenges and Solutions

Problem - Difficulty in Mathematical Development:

The initial approach was to develop a mathematical model for the language capacity. This mathematical development relied on combinatorial and probabilistic calculations (as recommended by the advisor). However, due to insufficient mathematical tools or the inability to achieve the desired calculations, it was decided to explore a different approach to understanding the language capacity.

Solution: It was decided to gain intuition about the language capacity through software simulations as detailed in Program 1.

Problem - Long Runtime for Simulations:

At the beginning of writing the first program, it was understood that the simulations were taking an impractically long time.

Solution: The code was optimized with a sophisticated algorithm to reduce runtime for small (k) values, as detailed earlier.

Reminder: The runtime in this approach is linear (O(n)) instead of (O(n*m*k)), which is a significant improvement. The program's runtime is reasonable even when running simulations with (n) on the order of (10^9), which would cause the naive algorithm to run for months.

Problem - General Difficulty in Writing the Algorithm:

The algorithm is a coding algorithm for a constraint that did not previously exist, requiring originality and creativity in its creation.

Solution: Similar algorithms from related problems (e.g., from the paper [1]) were hybridized. Inspiration was drawn from both simpler and more complex algorithms.

Problem - Error in the Expansion Part of the Algorithm:

Reminder: The algorithm is divided into three parts, with the third part being the expansion part. While writing the proof for the algorithm, a difficulty arose. Consultation with the project advisor revealed an error in the expansion part. The algorithm relied on a theorem that it did not meet the conditions for (there were not enough subwords to add without repetitions).

Solution: Correcting the error required extensive work, and due to time constraints, it was decided to proceed with the algorithm knowing there was an error in the expansion part. The reason for this decision is that the algorithm still provides coding that meets the project objectives (as previously mentioned, only affecting the length of the returned word).

Problem - Computer Memory Limitation:

Reminder: Our algorithm is valid for very large values (m) on the order of (log(n)), (k) on the order of (log(log(n))), and marker on the order of (log(log(log(n))). To implement the algorithm with realistic values that match the algorithm, very long words need to be created. For example, for a marker length of 3, (n) needs to be $n = 2^{2^{2^{marker}}} = 2^{2^{2^3}} = 2^{2^8} = 2^{256}$

This (n) is not feasible for any computer in the world.

Solution: Consequently, a "compromise" was made, and instead of strict values, the algorithm was relaxed (i.e., an algorithm with greater redundancy was used). The values set were (n = 65,536), (m = 256), (k = 18), and marker = 3. These values provide relevant content for the algorithm, sufficient repetitions, and enough margin for adjustment.

Problem - Need for Constrained Coding:

As mentioned above, the algorithm from the document relies on the existence of an algorithm for constrained coding to remove the marker from the repeat position coding. Implementing the algorithm required creating such coding, which was beyond the project's scope.

Solution: After careful consideration, the authors developed an algorithm to encode an 8-bit word into a 9-bit word without three consecutive zeros, meeting the algorithm's requirements (see Appendix B).

Problem - Error in the Implemented Algorithm:

As previously noted, there is an error in the implementation. This error was detected during text comparison (text diff). It occurs randomly every few thousand bits.

Solution: No solution was found for this problem.

9) Conclusions and recommendations

In summary, the project successfully provided a working algorithm for the presented problem. This project included three comprehensive computer programs that help understand the problem and demonstrate that the provided algorithm is practical and feasible. Additionally, an algorithm with a proof of correctness was supplied. The project delivered studied, engineered, and implemented solution to a research problem.

Recommendations for Further Work

- The expansion part of the algorithm needs to be completed.

- The encoding and decoding algorithms could be implemented for larger values of (n), (m), and (k). This would allow for more extensive simulations and help identify the implementation error. The implementation could use bits in the computer as a binary representation instead of using the char type. This would reduce runtime or increase the values by a factor of 8 without additional cost. This requires more complex implementation planning and was therefore not done.

- The algorithm could be extended to a general alphabet rather than just a binary one as in our implementation.

- A mathematical analysis could be conducted to theoretically calculate the language capacity, which was not successful in our case.

Our results align with the project objectives. An encoding algorithm was developed with minimal redundancy ($k = O(\log\log(n))$. Additionally, in principle, (k) can be reduced as desired as long as the relation ($k > \log(m)$) is maintained, which is a lower bound.

Advantages of the Approach

- Efficiency of the algorithm and minimal redundancy.

- Fast runtime of the computer programs.

- Innovation.

Disadvantages of the Approach

- The expansion part is missing.

- The solution is not general (only valid for a binary alphabet).

It is evident that the project objectives were fully achieved.

10) References

[1] O. Elishco, R. Gabrys, E. Yaakobi, and M. Medard, "Repeat-free codes," arXiv preprint arXiv:1909.05694, vol. 2, 2021.

11) Appendix

Appendix A - Academic Document and Proposed Algorithm:

“Repeat Free Codes with Windows Constraints.pdf”

Appendix B - Encoding algorithm:

“Constraint Algorithm For Project.pdf”

Appendix C - Computer Program for Sequence Generation and Validation:

“kmnRandomCheck.c”

Appendix D – Encoder implementation:

“CreateAndEliminate256183.c”

Appendix E – Decoder implementation:

“Decode256183.c”

Back to table of contents