Compressed Data Structures for DNA Sequences: CSA, CST, and Succinct Representations

Mann Kaniyawala
URA - Department of Computing and Software

Abstract

With increasing amounts of genomic data, especially the FASTQ DNA sequence files, there is a growing demand for efficient storage and querying methods. We have methods like GZIP which provide limited improvements, especially for repetitive genomic sequences. In this report, we look at other compressed data structures: Compressed Suffix Arrays (CSA), Compressed Suffix Trees (CST), along with Wavelet Trees and Block Trees. We will look at their definitions, implementations using GitHub repositories (SDSL and MinimalistBT-CST), and their potential for improving DNA sequence compression and analysis.

1 Definitions and Concepts

1.1 Compressed Suffix Array (CSA)

A suffix array stores all suffixes of a string in lexicographic order. A Compressed Suffix Array (CSA) achieves the same functionality but with much less space. It supports operations like substring search, locate, and extract without storing the full array.

Examples: `csa_sada`, `csa_wt`
Usage in Genomics: Ideal for locating motifs or subsequences.
Applications:

- String matching in genome indexing.
- Efficient storage for reference-based compression.
- Pattern locating in large-scale bioinformatics pipelines.

1.2 Compressed Suffix Tree (CST)

A suffix tree is a compact trie of all suffixes. CSTs simulate this structure in a compressed format, integrating LCP array, CSA and navigation helpers. This structure supports queries like Lowest Common Ancestor (LCA), parent-child traversal and pattern matching.

Examples: `cst_sada`, `cst_cst3`
Applications:

- Structural variant detection.
- Repeat finding and motif discovery.
- Range minimum queries and alignment operations.

1.3 Wavelet Tree

A wavelet tree is a binary tree used to represent strings efficiently with support for `select`, `rank`, and `access` operations.

Applications:

- Internally used in `csa_wt` for compact encoding.
- Used in genomic k-mer indexing.
- Fast rank/select operations on compressed sequences.

1.4 Block Tree

A block tree recursively compresses strings by identifying and representing repeated substrings as blocks. This method is efficient for large, repetitive datasets like genomes.

Applications:

- Used in MinimalistBT-CST for grammar-compressed CSTs.
- Efficient full-text indexing on highly repetitive data.
- Compressed representation of pan-genomes.

2 Setup and Code

2.1 SDSL Library

GitHub: `https://github.com/simongog/sdsl`

Listing 1: CSA Construction using SDSL

```
#include <sdsl/suffix_arrays.hpp>
#include <iostream>
#include <fstream>

using namespace sdsl;

int main() {
    std::string seq = "ACTGACTGACTG..."; // up to 10,000 bases
    std::ofstream out("dna.txt"); out << seq; out.close();

    csa_wt<> csa;
    construct(csa, "dna.txt", 1);
    store_to_file(csa, "dna_csa_wt.sdsl");

    std::cout << "Suffix at index 100: " << csa[100] << std::endl;
    return 0;
}
```

Replace `csa_wt<>` with `csa_sada<>`, `cst_sada<>`, or `cst_sct3<>` to try other types. Use `load_from_file()` for quick reuse.

2.2 MinimalistBT-CST

GitHub: `https://github.com/elarielcl/MinimalistBT-CST`

Listing 2: MinimalistBT-CST Setup

```
git clone https://github.com/elarielcl/MinimalistBT-CST.git
cd MinimalistBT-CST
make
./main data/your_sequence.fa
```

Features:

- Recursive grammar compression for memory efficiency.
- Faster navigation than classic suffix trees.
- Suited for highly repetitive genomic data.

3 SDSL vs MinimalistBT-CST Comparison

Table 1: Comparison between SDSL and MinimalistBT-CST

Feature	**SDSL**	**MinimalistBT-CST**
Purpose	General-purpose	Genomic repetition-focused
Compression Technique	Wavelet Tree (Entropy-based)	Block Tree (Grammar-based)
Performance	Good for varied input	Excellent for repetitive input
Core Structures	CSA, CST, Wavelet Tree	CST with block grammar
Code Complexity	High	Modular and Lightweight

References

[1] S. Gog. *SDSL GitHub Repository.*
`https://github.com/simongog/sdsl`

[2] E. L. Claude. *MinimalistBT-CST GitHub Repository.*
`https://github.com/elarielcl/MinimalistBT-CST`

[3] G. Navarro, A. Ordóñez, et al. (2021). Minimalist and Efficient Compressed Suffix Trees for Repetitive Texts.
Information Processing & Management, 58(2), 102520.
`https://www.sciencedirect.com/science/article/pii/S089054012100064X`

[4] V. Makinen and G. Navarro. (2005). Succinct suffix arrays based on run-length encoding. In *CPM 2005: Combinatorial Pattern Matching.*