Constructing the CDAWG CFG using LCP-Intervals

Alan Cleary and Jordan Dood

Some Definitions

CFG - Context Free Grammar

CDAWG - Compacted Directed Acyclic Word Graph

LCP - Longest Common Prefix

Context Free Grammars

(CFGs)

- A CFG is a set of production rules
- Recursing the rules from a designated ‘start’ rule yields a string or language of strings
- Grammars that produce a single string are called ‘straight-line’ grammars

Why We Are Interested in CFGs

- CFGs can be built from repeated sub-sequences, and thus can result in compression, of the string
- CFGs are self-indexing
- CFGs can be constructed in a number of different ways, which offers flexibility

mississippi

S -> mXXiZi
X -> iY
Y -> ss
Z -> pp

(Pictographic representation)

Motivation

What is needed

We wanted to build a ’bridge’ between the CFG realm and that of other well studied string data structures

Putting It Another Way

We want to find a way of compressing strings as a context free grammars that …

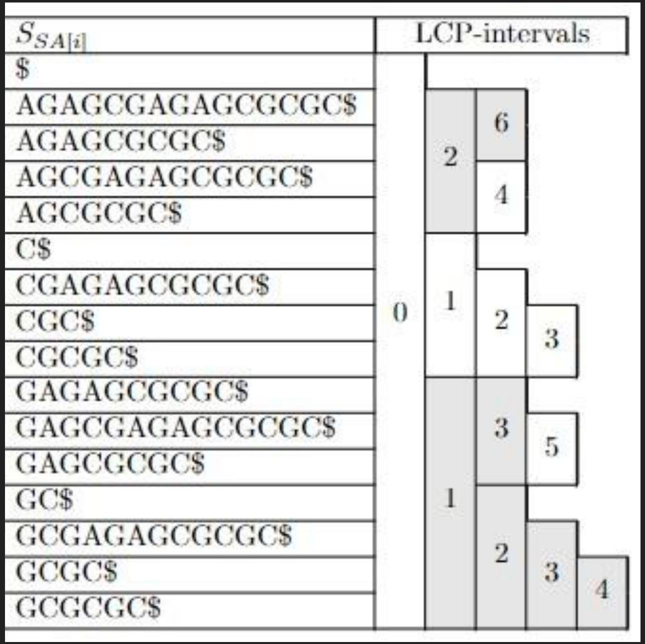
maintains the information structure related to other data structures, like suffix trees, CDAWGS, ect

and …

does so efficiently in both time and space, ideally *O(n)* time where *n* is the size of the string

Design Criteria

1. Compress data into a straight-line CFG
2. Build that CFG using the maximal repeats of the string
3. With a time complexity of *O(nlogn)* or *O(n)*



Repeats

A repeat is substring that occurs more than once in a string

A right-maximal repeat is repeat that cannot be extended further to the right without reducing the number of occurrences

A left-maximal repeat is is a repeat that cannot be extended further to the left without reducing the number of occurrences

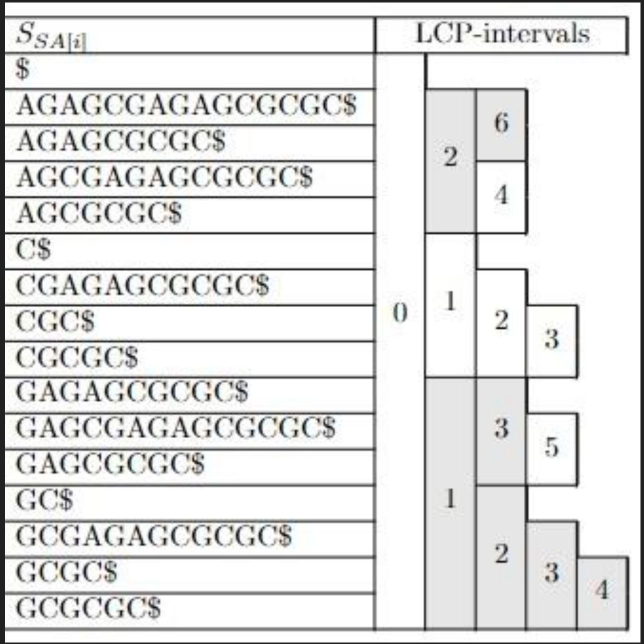
A maximal repeat is a repeat that is both right- and left-maximal and is a subset of both sets

Repeats

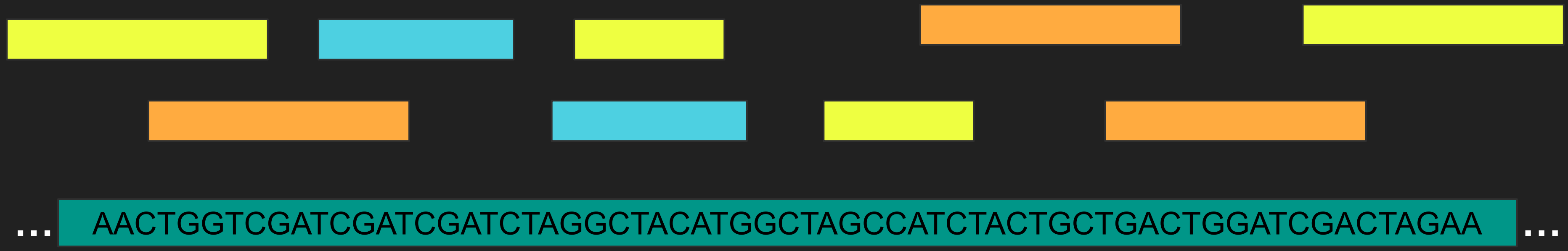
These repeats are inherent to a given string

Right-maximal repeats are represented by the internal nodes of both the suffix tree and the CDAWG, and define the Longest Common Prefix (LCP) intervals of the suffix array, the intervals of a suffix array that share a common prefix

Because of this connection to these data structures we wanted to utilize maximal repeats to build our grammar



The Crux



... AACTGGTCGATCGATCGATCTAGGCTACATGGCTAGCCATCTACTGCTGACTGGATCGACTAGAA ...

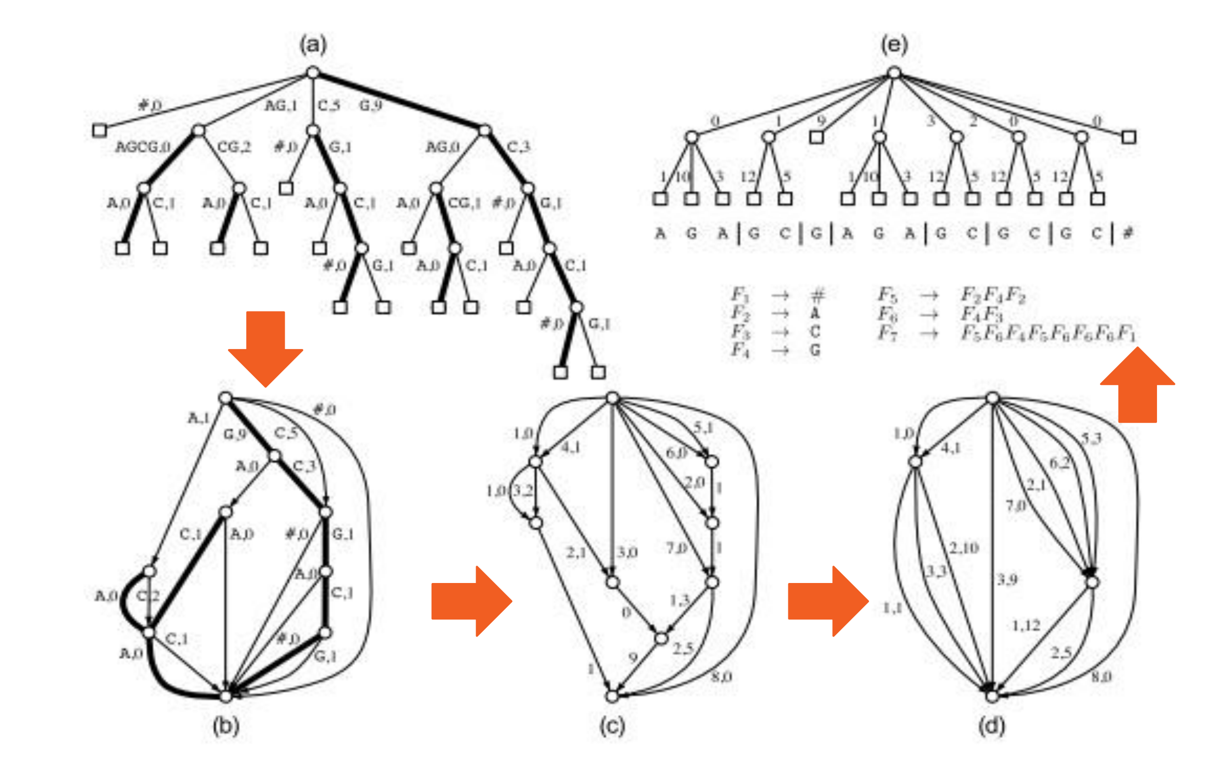
The Crux

The difficulty is in choosing which rules to include
and how to handle overlaps

Strait-line Grammar from Suffix Tree

This process uses a Compacted Directed Acyclic Word Graph (CDAWG)

As well as a number of simplification steps, to build a CFG from the structure of the CDAWG



Belazzougui, Djamal, and Fabio Cunial. "Representing the suffix tree with the CDAWG." 28th Annual Symposium on Combinatorial Pattern Matching (2017).

Key Inspiration

For CDAWG-CFG

Because there is a bijection between the internal nodes of the suffix tree and the right-maximal repeats, and because a subset of these nodes are preserved during the transformations, the grammar that results is comprised of these repeats.

We exploit this by noting the bijection between the internal nodes of a CDAWG and maximal repeats.

Why This Matters to Us

- Maximal repeats are (by definition) a subset of the right-maximal repeats, thus maintaining the relatedness to other data structures

- Since the Belazzougui, et al. work shows that right-maximal repeats can be used to build a grammar, and crucially, how these repeats are used to encode the target string, maximal repeats should be able to do something similar

- The downside is that this starts with a suffix tree and has a lot of intermediate graph structures

Comparison of Algorithms

Belazzougui, et al.

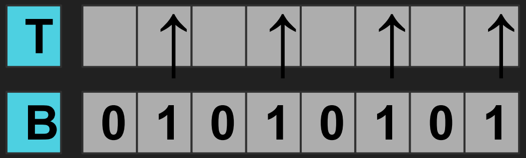
- Requires precomputting a suffix tree and CDAWG
- Uses CDAWG nodes (right-maximal repeats)
- Uses extraneous (for our purpose) steps

Our Algorithm

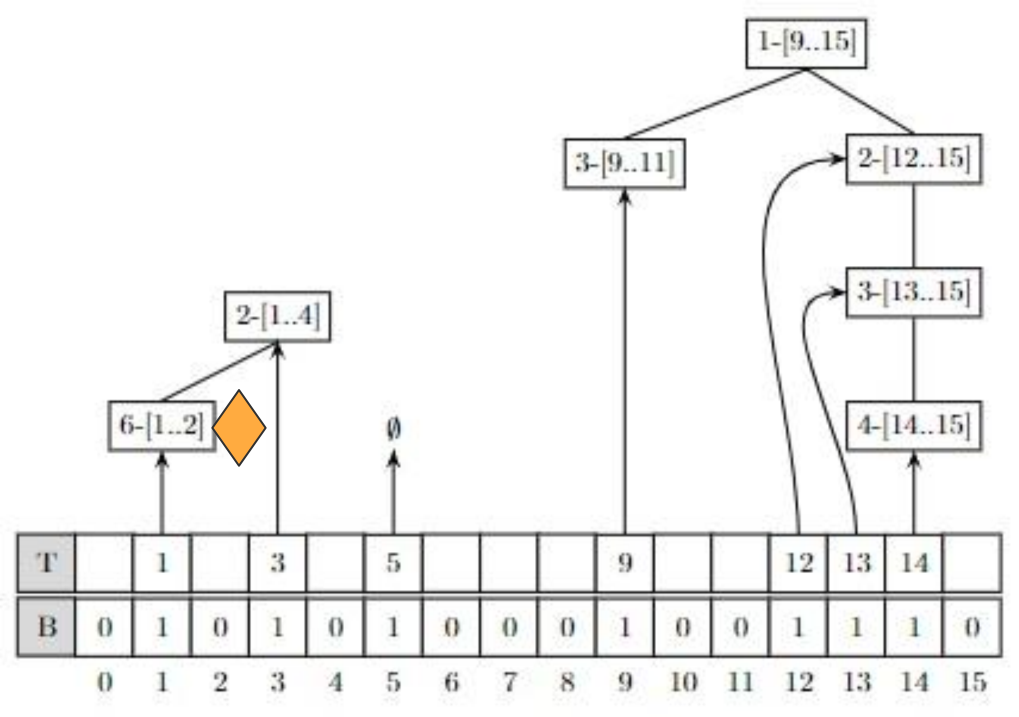
- Uses LCP-intervals, which can be computed from many data structures
- Can be computed online
- Computes the CFG directly

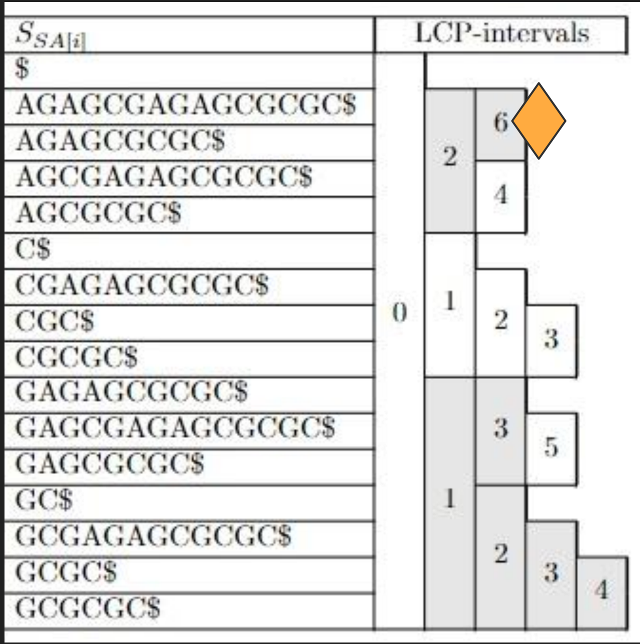
Our Approach

This is done by iterating the LCP-intervals and specifically uses those of maximal repeats to build the strait-line grammar



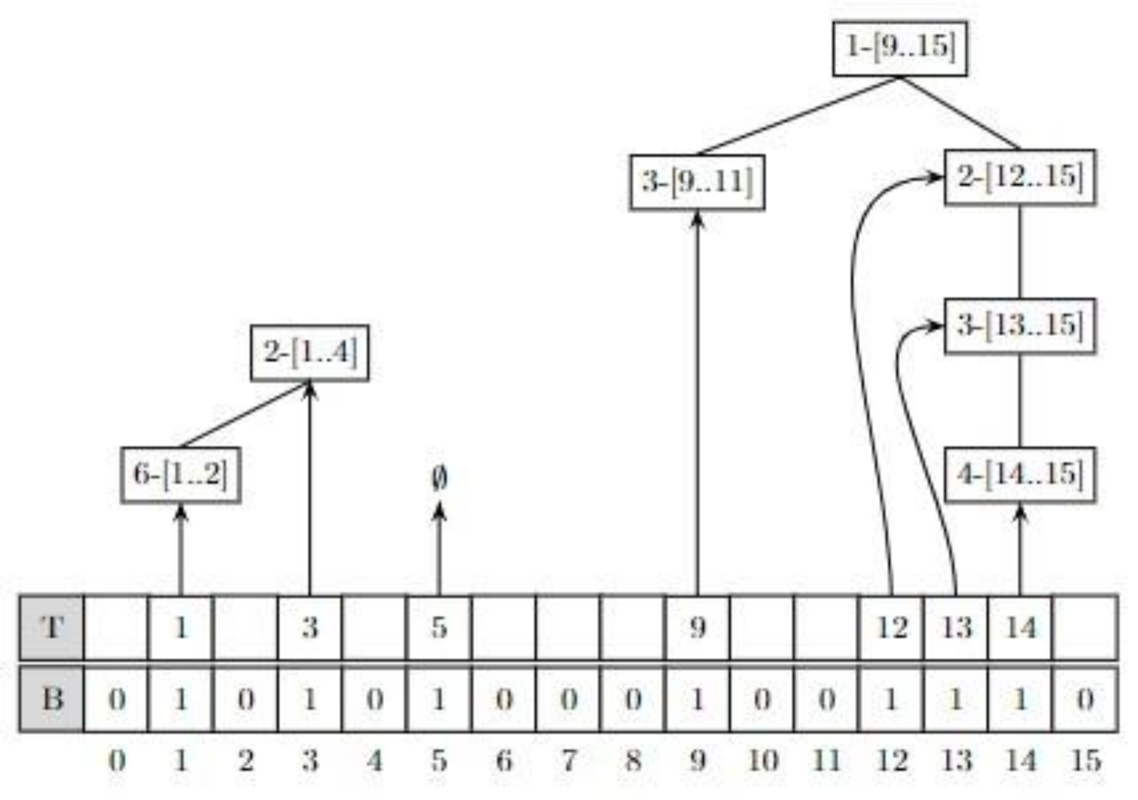
Our Data Structure





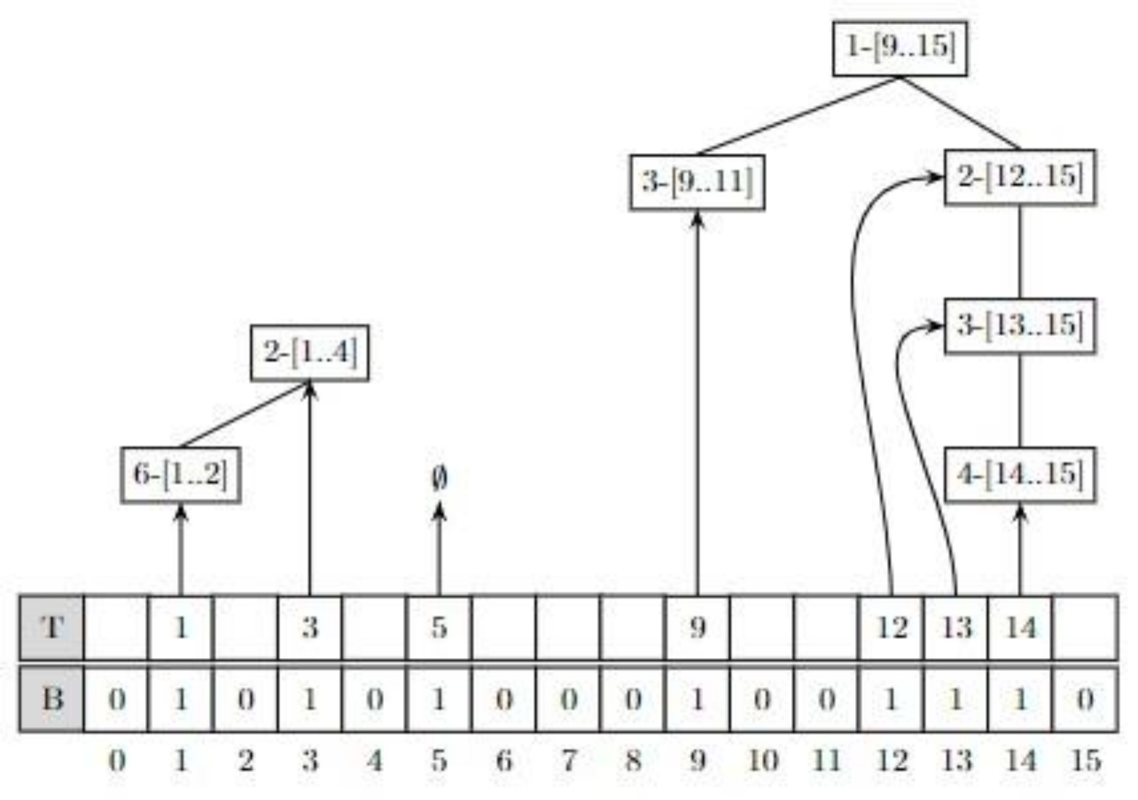
Our data structure is an interval tree for answering stabbing queries on LCP-intervals in constant time, or more generally nested intervals in a finite integer range

Our Data Structure



<- Pointer Map

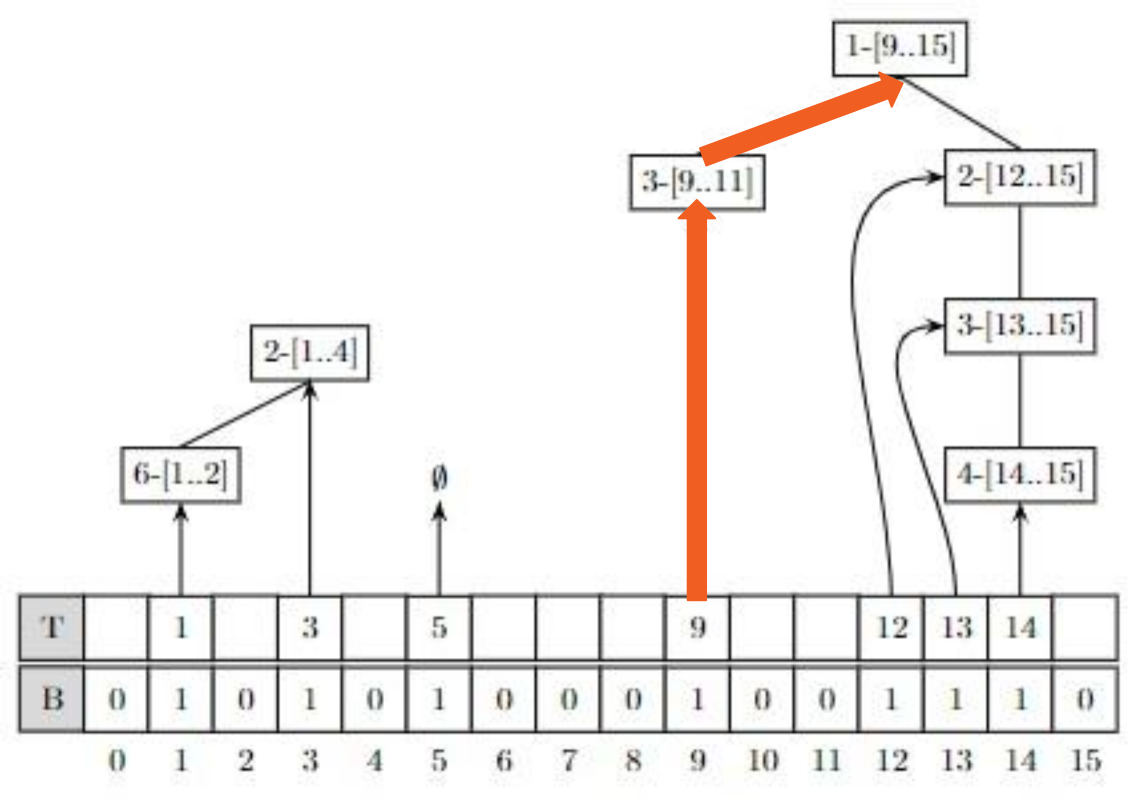
Our Data Structure

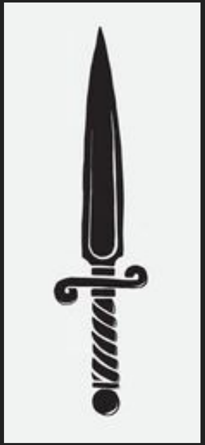


<- Pointer Map

<- Bit Vector

Our Data Structure





https://www.shutterstock.com/search/dagger-silhouette

2-Modes of Action

Our data structure can be built first, then preprocessed in *O(n)* so that 'stabbing' is done in *O(1)* …

Or it can be used online which allows for stabbing queries to be done in *O(log n)* time (binary search)

This duality is possible because of the nested structure of LCP-intervals and the order of their traversal

Preprocessed *O(n) time*

Iterate LCP-intervals, Build Data Structure → Preprocess Data Structure → Iterate LCP-intervals, and Encode Grammar → Encode Start

Online *O(n log n) time*

Iterate LCP-intervals, Building Data Structure and Encoding Grammar → Encode Start

Our Algorithm

(Optimal)

1) Iterate the LCP-intervals

AATCCTCATCGTCCATG …

CCATT

Maximal?

Beller, Timo, Katharina Berger, and Enno Ohlebusch. "Space-efficient computation of maximal and supermaximal repeats in genome sequences." *International Symposium on String Processing and Information Retrieval*. Springer, Berlin, Heidelberg, 2012.

1) Iterate the LCP-intervals

AATCCTCATCGTCCATG ...

CCATT Maximal?

The iteration is done in first length, then lexicographic order

Beller, Timo, Katharina Berger, and Enno Ohlebusch. "Space-efficient computation of maximal and supermaximal repeats in genome sequences." *International Symposium on String Processing and Information Retrieval*. Springer, Berlin, Heidelberg, 2012.

1) Iterate the LCP-intervals

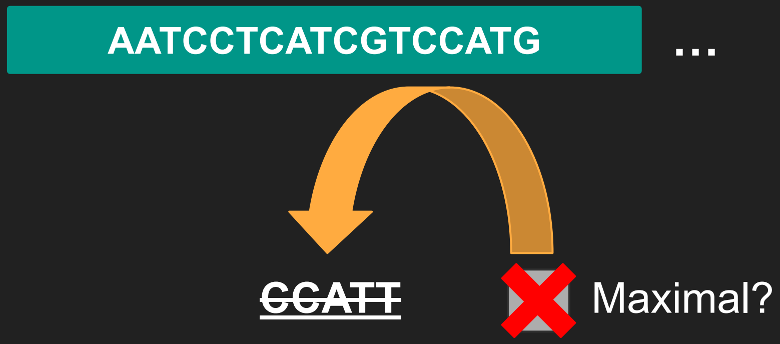
AATCCTCATCGTCCATG ...

CCATT ☐ Maximal?

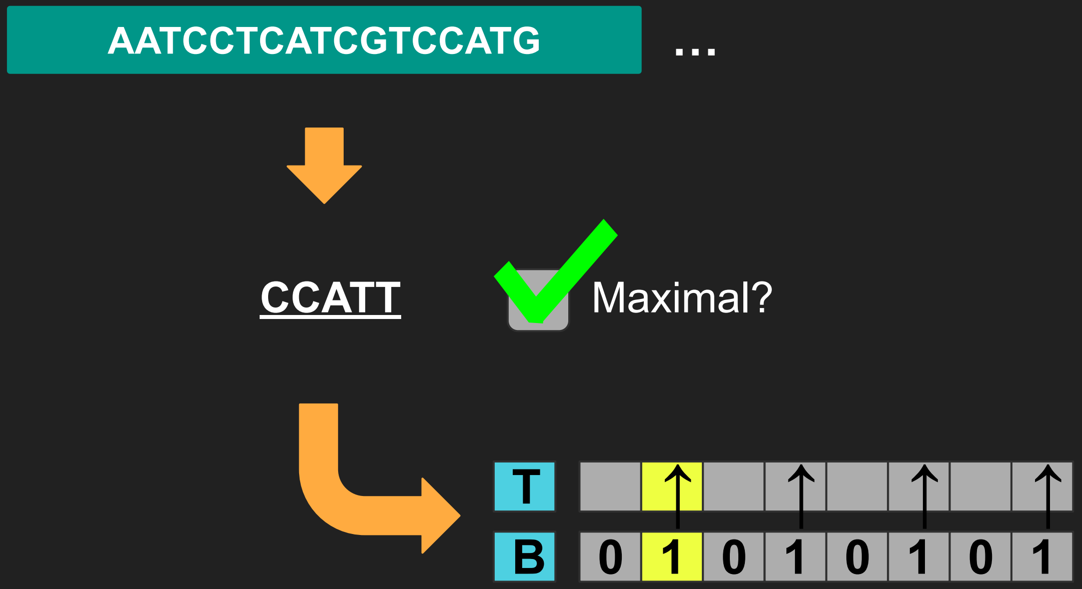
NOTE: LCP-intervals can be computed from a variety of string data structures, this one uses an FM-index

Beller, Timo, Katharina Berger, and Enno Ohlebusch. "Space-efficient computation of maximal and supermaximal repeats in genome sequences." *International Symposium on String Processing and Information Retrieval*. Springer, Berlin, Heidelberg, 2012.

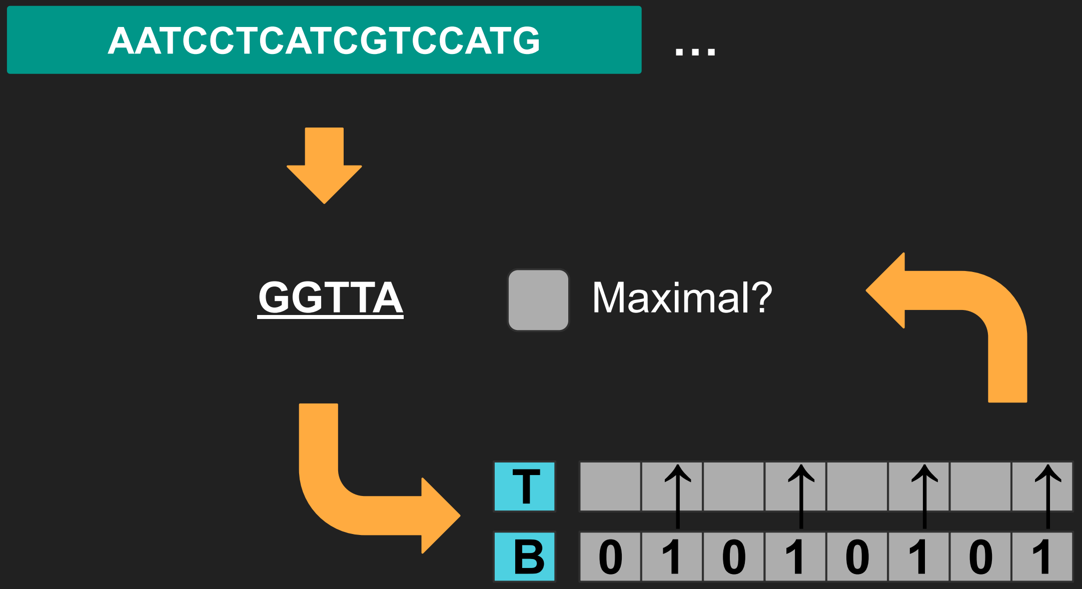
2) Determine Maximality



3) Add to Stabbing Data Structure

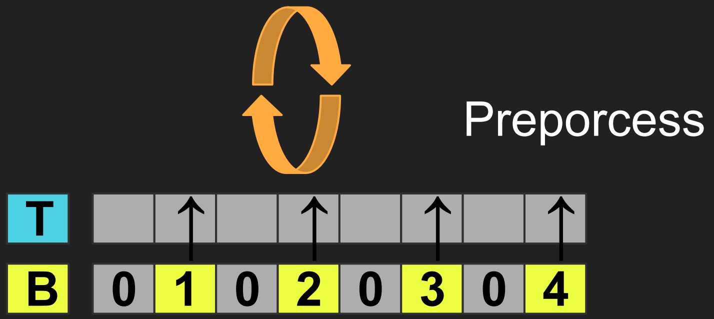


4) Repeat (Steps 1-3)

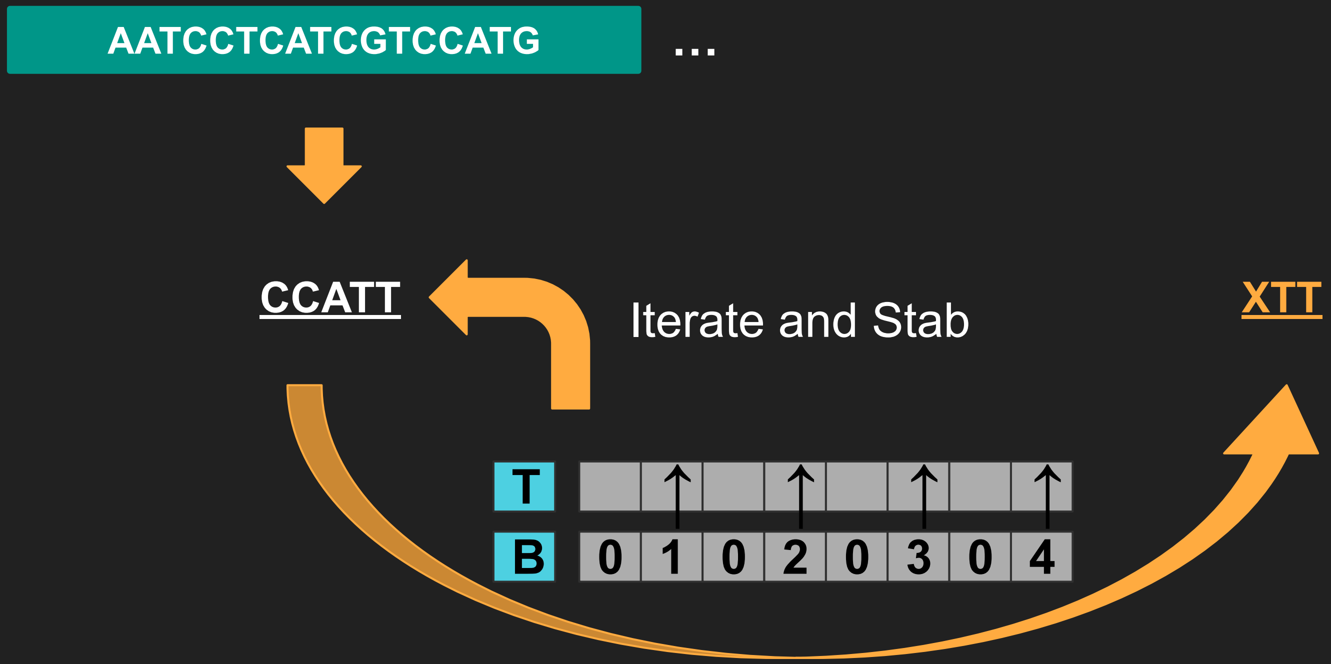


5) Process The Stabbing Data Structure

AATCCTCATCGTCCATG ...



6) Encode The Rules



Where **X = CCA** (a previous rule) found by stabbing, that prefixes the current rule

6) Encode The Rules

7) Repeat (step 6)

8) Encode The Start Rule

Our Implementation

(and some results)

https://github.com/alancleary/cdawg-cfg

Conclusions

Our Algorithm . . .

- Compresses data into a CFG that is based on maximal repeats
- This can be done online or not, with time complexity of $O(n \log n)$ or $O(n)$, respectively
- We achieve this using a novel interval stabbing data structure to reduce intermediate steps and data structures

Future Work

- Use the relation to other string data structures to port over functionality
- Better characterize and improve the compression ratio for our grammars
- Explore the opportunities for operating on compressed data

Funding and Acknowledgements

This work was supported by NSF Award Number 2105391

NCGR
National Center for Genome Resources

MONTANA STATE UNIVERSITY